SPERRY UNIVAC Series 1100 Executive System

Volume 2
EXEC Level 36R2

Programmer Reference

SPERRY UNIVAC COMPUTER SYSTEMS

UP-4144.23

This document contains the latest information available at the time of publication. However, Sperry Univac reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Sperry Univac representative.

Sperry Univac is a division of Sperry Corporation.

PRINTED IN U.S.A.

Page Status Summary

Issue: UP-4144.23 Update A

Section	Pages	Update
Cover/Disclaimer		
PSS	1	A
Preface	1,2	A
Contents	1 thru 15	Orig.
1	1 thru 10	Orig.
2	1 thru 27	Orig.
3	1 thru 75	Orig.
4	1 thru 90	Orig.
5	1 thru 31	Orig.
6	1 thru 55	Orig.
7	1 thru 28	Orig.
8	1 thru 44	Orig.
9	1 thru 37	Orig.
10	1 thru 12	Orig.
11	1 thru 14	Orig.
12	1 thru 26	Orig.
Appendix A	1 thru 11	Orig.
Appendix B	1 thru 9	Orig.
Appendix C	1 thru 38	Orig.
Appendix D	1 thru 7 8 thru 11 12 thru 19	Orig. A Orig.
Appendix E	1, 2	Orig.
Appendix F	1 thru 18	Orig.
Index	1 thru 22	Orig.

Section	Pages	Update
User Comment Sheet		
Total: 588 pages and cover		

Section	Pages	Update

Preface

The SPERRY UNIVAC Series 1100 Executive System Programmer Reference manual has been divided into four volumes. These volumes are titled as follows:

1. SPERRY UNIVAC Series 1100 Executive System, Volume 1, Index, Programmer Reference.

   Volume 1 references terms and subjects covered in the other three volumes.

   a. UP-4144.1 provides a consolidated index for UP-4144.2, UP-4144.3, and UP-4144.4, the three volumes intended for use with EXEC level 32R1 and associated system processors and system utility program.

   b. UP-4144.11 provides a consolidated index for UP-4144.21, UP-4144.31, and UP-4144.41, the three volumes intended for use with EXEC level 33R1 and associated system processors and system utility programs.

   c. The index for Volume 2 intended for use with EXEC level 36R2 is included in UP-4144.22.

2. SPERRY UNIVAC Series 1100 Executive System, Volume 2, EXEC, Programmer Reference.

   Volume 2 describes the overall control of SPERRY UNIVAC Series 1100 Systems by the Executive System.

   a. UP-4144.21 is the programmer reference for EXEC level 33R1.

   b. UP-4144.21-A provides corrections for UP-4144.21.

   c. UP-4144.21-B upgrades UP-4144.21 to correspond to EXEC level 33R2.

   d. UP-4144.21-C provides additional changes for UP-4144.21 to correspond to EXEC level 33R2.

   e. UP-4144.21-D upgrades UP-4144.21 to correspond to EXEC level 33R3.

   f. UP-4144.22 is the programmer reference for EXEC level 35R1.

   g. PUP-4144.2P2 is a preliminary programmer reference for EXEC level 36R1.

   h. UP-4144.23 is the programmer reference for EXEC level 36R2.

3. SPERRY UNIVAC 1100 Series Executive System, Volume 3, System Processors, Programmer Reference.

   Volume 3 describes the basic system processors.

   a. UP-4144.31 describes the system processors associated with EXEC level 33R1.

   b. The addition of Update Package A (UP-4144.31-A) to UP-4144.31 produces a manual which describes the system processors associated with EXEC level 35R1.

   c. Programmer reference information for a number of the system processors described in Volume 3 has been released in separate UP manuals. These manuals provide either information on more recent levels of the system processors described in Volume 3 or upgraded information for the same levels described in Volume 3. These manuals are as follows:

      - UP-8721 on Collector (MAP Processor) level 29R1.
      - UP-8723 on Text Editor (ED Processor) level 15R2.
      - UP-8724 on FURPUR level 27R3.
      - UP-8726 on File Administration Processor (SECURE) level 20R1.
      - UP-8727 on Symbolic Stream Generator (SSG) level 18R1.

4. SPERRY UNIVAC 1100 Series Executive System, Volume 4, System Utility Programs, Programmer Reference.

   Volume 4 describes the System Relocatable Library (SYSLIB), system common banks, and utility processors.

   a. UP-4144.41 describes the system utility programs associated with EXEC levels 33R1 and 35R1. It includes information on SYSLIB level 73R1.

   b. Programmer reference information for more recent levels of the System Relocatable Library and Common Banks software is being released in a series of separate manuals, independent of Volume 4. The first of these is:

      - UP-8728 on System Relocatable Library and Common Bank level 74R1.

Contents

Page Status Summary

Preface

Contents

1. Introduction 1-1

1.1. SCOPE OF MANUAL 1-1

1.2. THE OPERATING SYSTEM 1-1

1.3. THE EXECUTIVE SYSTEM (EXEC 8) 1-2
1.3.1. Multiple Modes of Operation 1-2
1.3.1.1. Batch Processing 1-3
1.3.1.2. Demand Processing (Time Sharing) 1-3
1.3.1.3. Real-Time Processing 1-3
1.3.1.4. Transaction Processing Interface 1-3
1.3.1.5. Multiprogramming and Multiprocessing 1-3
1.3.2. Utilization of Mass Storage 1-3
1.3.3. Functional Areas of the Executive System 1-4
1.3.3.1. Executive Control Language 1-4
1.3.3.2. The Supervisor 1-4
1.3.3.3. Facilities Assignment 1-5
1.3.3.4. File Control 1-5
1.3.3.5. Operator Communications 1-5
1.3.3.6. Input/Output Device Handlers and Symbionts 1-5

1.4. SYSTEM PROCESSORS 1-7
1.4.1. Collector (MAP) 1-7
1.4.2. File Utility Processor (FURPUR) 1-7
1.4.3. Postmortem Dump Processor (PMD) 1-8
1.4.4. DATA Processor 1-8
1.4.5. ELT Processor 1-8
1.4.6. File Administration Processor (SECURE) 1-8
1.4.7. Text Editor (ED) 1-8
1.4.8. Procedure Definition Processor (PDP) 1-8
1.4.9. SSG Processor 1-8
1.4.10. Conversational Time Sharing (CTS) 1-8
1.4.11. High Volume Time Sharing (HVTS) 1-9

1.5. SYSTEM UTILITY ROUTINES 1-9
1.5.1. CULL Processor 1-9
1.5.2. Document Processor (DQC) 1-9
1.5.3. LIST Processor 1-9
1.5.4. 1100 Virtual Machine Control Processor (FLIT) 1-9

1.6. LANGUAGE PROCESSORS 1-9

1.7. RELOCATABLE SUBROUTINE LIBRARY 1-10

1.8. APPLICATIONS PROGRAMS 1-10

2. General Concepts and Definitions 2-1

2.1. INTRODUCTION 2-1

2.2. DEFINITIONS AND ABBREVIATIONS 2-1
2.2.1. Definitions 2-1
2.2.2. Abbreviations 2-13

2.3. CONVENTIONS 2-15
2.3.1. Notational Conventions 2-15
2.3.2. Control Statement Notation 2-16

2.4. BASIC CONCEPTS OF RUN CONTROL 2-17
2.4.1. Run Initiation 2-17
2.4.2. Run Execution 2-17
2.4.3. Symbiont File Concepts 2-17
2.4.4. Run Termination 2-18

2.5. BASIC CONCEPTS OF TASK CONTROL 2-19
2.5.1. Real-Time Mode 2-19
2.5.2. Task Initiation 2-19
2.5.3. Task Execution and Switching 2-19
2.5.4. Executive Requests 2-20
2.5.5. Multiprogramming Considerations 2-20
2.5.6. Task Termination 2-21
2.5.7. Program Protection 2-21
2.5.8. Address Space Sharing 2-22

2.6. FILENAMES AND ELEMENT NAMES 2-22
2.6.1. Filenames 2-22
2.6.2. External and Internal Filenames 2-23
2.6.3. File Cycles (F-Cycles) 2-24
2.6.4. Element Names 2-25
2.6.5. Symbolic Element Cycle 2-25
2.6.6. Referencing Files and Elements 2-26
2.6.7. Examples of File and Element Reference 2-27

3. Executive Control Statements 3-1

3.1. INTRODUCTION 3-1

3.2. CONTROL STATEMENT FORMAT 3-1
3.2.1. Label Field 3-2
3.2.2. Operation Fields 3-2
3.2.3. Operand Fields 3-2
3.2.4. Control Statement Annotation 3-2
3.2.5. Control Statement Continuation 3-3
3.2.6. Leading Blanks in Fields 3-3
3.2.7. General Default Rules 3-3
3.2.8. Transparent Control Statements 3-4

3.3. SUMMARY OF CONTROL STATEMENTS 3-4

3.4. SCHEDULING CONTROL STATEMENTS 3-7
3.4.1. Run Initiation (@RUN) 3-7
3.4.1.1. Postmortem and Dynamic Dump Options 3-11
3.4.1.2. Run Recovery (R Option) 3-12
3.4.1.3. Examples of @RUN Statements 3-12
3.4.2. Run Termination (@FIN) 3-13
3.4.3. Dynamic Initiation of an Independent Run (@START) 3-14
3.4.4. Initiating Execution (@XQT) 3-16
3.4.4.1. Initial Execution Status 3-16
3.4.4.2. Initial Program State 3-17
3.4.4.2.1. Overlapped Addresses 3-17
3.4.4.2.2. Lowest Bank Address 3-17
3.4.4.2.3. Initially-Based Common Banks 3-18
3.4.4.2.4. Common Bank Access 3-18
3.4.4.3. Program Data Separation (@EOF) 3-18
3.4.4.4. Bank Referencing 3-19
3.4.4.4.1. Visible Banks 3-19
3.4.4.4.2. Switching Between Banks (LBJ, LIJ, LDJ) 3-20
3.4.4.4.3. Static versus Dynamic Banks 3-21
3.4.4.4.4. Initial Load 3-22
3.4.4.4.5. Active Banks 3-22
3.4.4.4.6. Program Control Table (PCT) Referencing 3-22
3.4.4.4.7. Bank Unbasing 3-22
3.4.4.5. Program Termination 3-23

3.5. MESSAGE CONTROL STATEMENTS 3-23
3.5.1. Displaying a Message (@MSG) 3-23
3.5.2. Inserting Information in the Master Log (@LOG) 3-25

3.6. SYMBIONT DIRECTIVE STATEMENTS 3-25
3.6.1. Print Output Heading Control (@HDG) 3-26
3.6.2. Symbiont File Breakpointing (@BRKPT) 3-27
3.6.2.1. Primary Output File Breakpoint 3-27
3.6.2.2. Alternate Symbiont File Breakpoint 3-28
3.6.3. Symbiont Output File Queuing (@SYM) 3-29
3.6.4. @BRKPT/@SYM Control Statement Usage 3-31
3.6.5. Card Reader Mode Control (@COL) 3-32
3.6.6. Onsite C/SP,9000,0716 Card Reader Mode Control (@COL) 3-34

3.7. FACILITY CONTROL STATEMENTS 3-35
3.7.1. Assigning Files and Peripheral Devices (@ASG) 3-35
3.7.1.1. Sector-Formatted File Assignment 3-37
3.7.1.2. Magnetic Tape Assignment 3-45

3.7.1.3. Word-Addressable Mass Storage Assignment 3-52
3.7.1.3.1. Normal Assignment 3-52
3.7.1.3.2. Absolute Device Assignment 3-54
3.7.2. Tape Unit Mode Control (@MODE) 3-55
3.7.3. Independent Cataloging of Files (@CAT) 3-56
3.7.4. Releasing Files and Peripheral Devices (@FREE) 3-58
3.7.5. Attaching Internal Filenames (@USE) 3-61
3.7.6. Specifying Filename Qualifier (@QUAL) 3-62

3.8. DATA PREPARATION CONTROL STATEMENTS 3-63
3.8.1. Direct Creation of Card Image Files (@FILE) 3-63
3.8.2. Terminating the File Mode (@ENDF) 3-64
3.8.3. Direct Creation of Data Files from NTR Sites 3-64

3.9. PROCESSOR CONTROL STATEMENTS 3-65

3.10. DYNAMIC RUNSTREAM MODIFICATION 3-68
3.10.1. Dynamic Runstream Expansion (@ADD) 3-68
3.10.2. Conditional Statements 3-70
3.10.3. Statement Labeling 3-70
3.10.4. Condition Word 3-71
3.10.4.1. Condition Word Control (@SETC) 3-72
3.10.4.2. Condition Word Testing (@TEST) 3-73
3.10.4.3. Branching from within Runstream (@JUMP) 3-74
3.10.4.4. Conditional Runstream Example 3-75

4. Executive Service Requests 4-1

4.1. INTRODUCTION 4-1
4.1.1. Coding Restrictions 4-1
4.1.2. Calling Sequence Conventions 4-1
4.1.3. ER Synchrony 4-2
4.1.4. Error Handling 4-2

4.2. SUMMARY OF EXECUTIVE REQUESTS 4-3

4.3. ACTIVITY AND PROGRAM CONTROL 4-6
4.3.1. Activity Registration 4-6
4.3.1.1. Create a New Activity (FORK$) 4-6
4.3.1.2. Create a New Activity with Timed WAIT (TFORK$) 4-7
4.3.1.3. Activity/Processor Dedication (ADED$) 4-7
4.3.2. Activity Termination 4-8
4.3.2.1. Activity Normal Termination (EXIT$) 4-8
4.3.2.2. Activity Error Termination (ERR$) 4-8
4.3.2.3. Abort Run (ABORT$) 4-8
4.3.2.4. Run Error Termination (EABT$) 4-9
4.3.3. Activity Synchronization 4-9
4.3.3.1. Joining of Activities (AWAIT$) 4-10
4.3.3.2. Activity Naming (NAME$) 4-10
4.3.3.3. Activity Deactivation (DACT$) 4-10
4.3.3.4. Activity Activation (ACT$) 4-11
4.3.3.5. Inter-Activity Interrupt (INT$) 4-11
4.3.3.6. Activity Identification (IDENT$) 4-12
4.3.4. Test and Set Queuing 4-13
4.3.4.1. TSQ Cell Format (T$CELL) 4-14

4.3.4.2. TSQ Registration (TSQRG$) 4-14
4.3.4.3. TSQ Deregistration (TSQCL$) 4-15
4.3.4.4. Clear Test and Set and Notify Executive (C$TS) 4-15
4.3.4.5. Clear Test and Set and Queue (C$TSQ) 4-15
4.3.4.6. Clear Test and Set and Activate (C$TSA) 4-16
4.3.4.7. Examples of TSQ Usage 4-17
4.3.5. Real-Time Program/Activity Control 4-18
4.3.5.1. Changing Program/Activity to Real-Time Status (RT$) 4-18
4.3.5.2. Removal of Program/Activity Real-Time Status (NRT$) 4-19
4.3.6. Timed Activity Wait (TWAIT$) 4-19
4.3.7. User Activity Leveling (LEVEL$) 4-19

4.4. CONDITION WORD CONTROL 4-20
4.4.1. Setting the Condition Word (SETC$) 4-20
4.4.2. Condition Word Retrieval (COND$) 4-20

4.5. RETRIEVAL OF THE TIME AND DATE 4-21
4.5.1. Time and Date in Fieldata (DATE$) 4-21
4.5.2. Time and Date in Binary (TDATE$) 4-21
4.5.3. Time in Milliseconds (TIME$) 4-22

4.6. CONSOLE COMMUNICATIONS 4-22
4.6.1. Console Output and Solicited Input (COM$) 4-22
4.6.2. Wait for Unsolicited Console Input (II$) 4-24

4.7. PROGRAM STORAGE CONTROL 4-25
4.7.1. Main Storage Expansion (MCORE$) 4-25
4.7.2. Main Storage Contraction (LCORE$) 4-25
4.7.3. Restrictions on the Use of MCORE$/LCORE$ 4-26
4.7.4. Main Storage Absolute Addressing (ABSAD$) 4-27
4.7.5. Segment Loads (LOAD$) 4-28
4.7.6. Down by Track (BDSPT$) 4-29
4.7.7. Communicating with the SMU (SMU$) 4-32

4.8. PROGRAM/SYSTEM INFORMATION RETRIEVAL 4-34
4.8.1. Retrieving @XQT Control Statement Options (OPT$) 4-34
4.8.2. Program Control Table Retrieval (PCT$) 4-34
4.8.3. Master Configuration Table Retrieval (MCT$) 4-36
4.8.3.1. Packet and Call 4-36
4.8.3.2. Read Functions 4-37
4.8.3.3. Write Functions 4-38
4.8.3.4. Error Codes 4-39
4.8.4. Bank Descriptor Index Retrieval and Common Bank Reload (BANK$) 4-40
4.8.4.1. BDI Retrieval 4-40
4.8.4.2. Common Bank Reload 4-42
4.8.5. Software Instrumentation Package (SIP) Interface (SYSBAL$) 4-44
4.8.6. Jump Stack Editing (EDJS$) 4-48
4.8.7. Miscellaneous Information Retrieval (INFO$) 4-49
4.8.8. Create User-formatted Log Entry (LOG$) 4-53

4.9. CONTINGENCIES 4-54
4.9.1. Introduction 4-54
4.9.2. Contingency Types and Standard Action 4-54
4.9.2.1. Error Termination Considerations 4-55
4.9.3. Contingency Registration 4-59

4.9.3.1. IALL$ 4-59
4.9.3.2. CREG$ 4-61
4.9.4. Contingency Processing (Non-ESI) 4-62
4.9.4.1. Contingency Routine 4-62
4.9.4.2. Contingency Mode Termination (CEND$) 4-63
4.9.4.3. Contingency Mode Termination and Return (CRTN$) 4-63
4.9.4.4. Additional Contingency Considerations 4-64
4.9.4.5. Abort Contingency 4-65
4.9.4.6. Hardware Fault Contingency 4-65
4.9.5. ESI Contingencies 4-65
4.9.6. Common Data Bank (CDB) Contingencies 4-67
4.9.6.1. CDB Contingency Registration 4-68
4.9.6.2. CDB Contingency Processing 4-68
4.9.6.3. Queuing Contingencies for the User Program (CQUE$) 4-69
4.9.6.4. Abnormal Program Termination while Executing within Common Banks 4-70
4.9.6.5. Notification of Common Banks of Activity Termination (TRMRG$) 4-71

4.10. MISCELLANEOUS EXECUTIVE REQUESTS 4-73
4.10.1. Dynamic Request of Control Statements 4-73
4.10.1.1. Request with Fieldata Images (CSF$) 4-73
4.10.1.2. Request with ASCII Image (ACSF$) 4-75
4.10.2. Retrieving Information from and Altering the Processor State Register 4-75
4.10.2.1. Store Processor Designators (SPD) 4-75
4.10.2.2. Load Processor Designators (LPD) 4-76
4.10.2.3. Executive Request PSR$ 4-77
4.10.3. Main Storage Snapshot Dump (SNAP$) 4-78
4.10.4. Setting User Breakpoint (SETBP$) 4-79
4.10.5. Error Message Printout (ERRPR$) 4-81

4.11. DYNAMIC AND COMMON BANK USAGE 4-84
4.11.1. Usage of Multiple Banks 4-84
4.11.1.1. Common Banks 4-84
4.11.1.1.1. Common Bank Reload 4-85
4.11.1.1.2. Non Configured Common Banks 4-85
4.11.1.2. Additional Instruction and Data Space 4-86
4.11.2. Write Protect Mode 4-87
4.11.3. Inter-Bank Addressing 4-87
4.11.3.1. Collection 4-87
4.11.3.2. Register Basing 4-88
4.11.3.3. Collector Produced Tables 4-88
4.11.4. Bank Address Limits 4-88
4.11.5. Executive Requests within Common Banks 4-88
4.11.5.1. MCORE$ and LCORE$ Usage 4-88
4.11.5.2. Contingency Registration and Processing 4-88
4.11.5.3. CMS$ and CPOOL$ Usage 4-89
4.11.5.4. LOAD$ Usage 4-89
4.11.5.5. IO$ Request 4-89
4.11.6. Dumping Common Banks 4-89
4.11.7. Data Protection and Activity Synchronization 4-89
4.11.7.1. TS Usage 4-89
4.11.7.2. ACT$, DACT$, AWAIT$ Synchronization 4-89
4.11.7.3. TS Queuing 4-89

5. Symbiont Interface Requests 5-1

5.1. INTRODUCTION 5-1
5.1.1. Symbionts 5-1
5.1.2. Symbiont/User Interface Routines 5-2

5.2. OBTAINING INPUT IMAGES 5-3
5.2.1. Reading Fieldata Images (READ$) 5-3
5.2.2. Reading ASCII Images (AREAD$) 5-4
5.2.3. Fieldata Images – Alternate File (READA$) 5-5
5.2.4. ASCII Images – Alternate File (AREADA$) 5-6
5.2.5. Fieldata Images – Conversational Mode (TREAD$) 5-7
5.2.6. ASCII Images – Conversational Mode (ATREAD$) 5-8

5.3. TRANSFERRING OUTPUT IMAGES 5-8
5.3.1. Printing Fieldata Images (PRINT$) 5-8
5.3.2. Printing ASCII Images (APRINT$) 5-9
5.3.3. Fieldata Images – Alternate Print File (PRNTA$) 5-9
5.3.4. ASCII Images – Alternate Print File (APRNTA$) 5-10
5.3.5. Punching Fieldata Images (PUNCH$) 5-11
5.3.6. Punching ASCII Images (APUNCH$) 5-12
5.3.7. Fieldata Images – Alternate Punch File (PNCHA$) 5-12
5.3.8. ASCII Images – Alternate Punch File (APNCHA$) 5-13

5.4. OUTPUT CONTROL FUNCTIONS 5-14
5.4.1. Fieldata Control Functions – Print File (PRTCN$) 5-14
5.4.2. ASCII Control Functions – Print File (APRTCN$) 5-19
5.4.3. Fieldata Control Function – Alternate Print File (PRTCA$) 5-19
5.4.4. ASCII Control Functions – Alternate Print File (APRTCA$) 5-20
5.4.5. Fieldata Control Functions – Punch File (PCHCN$) 5-20
5.4.6. ASCII Control Function – Punch File (APCHCN$) 5-21
5.4.7. Fieldata Control Functions – Alternate Punch File (PCHCA$) 5-22
5.4.8. ASCII Control Function – Alternate Punch File (APCHCA$) 5-22

5.5. CONTROL STATEMENT LISTING (CLIST$) 5-23

5.6. GENERAL PACKET-DRIVEN ER INTERFACE – SYMB$ 5-25

5.7. FIELDATA AND ASCII TRANSLATION 5-30

5.8. SYMBIONT OUTPUT FILE ERROR RECOVERY 5-30

6. Input/Output Device Interfaces 6-1

6.1. INTRODUCTION 6-1
6.1.1. Basic I/O Executive Request 6-1
6.1.2. Interrupt Activity 6-5
6.1.3. Queuing 6-5

6.2. I/O PACKET GENERATION 6-6
6.2.1. Magnetic Tape I/O Packet Generation (I$OT) 6-6
6.2.1.1. Magnetic Tape I/O Function without Interrupt 6-6
6.2.1.2. Magnetic Tape I/O Function with Interrupt 6-7
6.2.2. Mass Storage I/O Packet Generation (I$OD) 6-8
6.2.2.1. Mass Storage I/O Function without Interrupt 6-8

6.2.2.2. Mass Storage I/O Function with Interrupt 6-9

6.3. PROGRAM – I/O SYNCHRONIZATION 6-9
6.3.1. Wait for Completion of Specific I/O (WAIT$) 6-9
6.3.2. Wait for Completion of any I/O (WANY$) 6-10
6.3.3. Wait for Completion of any I/O (WALL$) 6-10
6.3.4. Initiate I/O and Return Control Immediately (IO$) 6-10
6.3.5. Initiate I/O and Return Control Immediately, with Interrupt (IOI$) 6-11
6.3.6. Initiate I/O and Wait for Completion (IOW$) 6-11
6.3.7. Initiate I/O and Wait for Completion, with Interrupt (IOWI$) 6-11
6.3.8. Initiate I/O and Exit, with Interrupt (IOXI$) 6-12
6.3.9. Reducing Interrupt Activity Priority (UNLCK$) 6-12

6.4. MAGNETIC TAPE HANDLER 6-13
6.4.1. Tape Handler Functions 6-13
6.4.1.1. Set Mode Function 6-15
6.4.1.2. Mode Set Function 6-18
6.4.2. General Considerations 6-20
6.4.2.1. Noise Constant Convention 6-20
6.4.2.2. Read Backward Limitations 6-20
6.4.2.3. Write Considerations 6-20
6.4.2.4. Move Considerations 6-21
6.4.2.5. Abnormal Frame Count Considerations 6-23
6.4.2.6. Scatter-Read/Gather-Write Considerations for High Speed Tapes 6-23

6.5. WORD-ADDRESSABLE FORMAT MASS STORAGE 6-26
6.5.1. Word-addressable Format Functions 6-26
6.5.2. General Considerations 6-26
6.5.3. EACQ$ Function 6-29

6.6. SECTOR-FORMATTED MASS STORAGE 6-32

6.7. DISK MASS STORAGE 6-35

6.8. ARBITRARY DEVICE INTERFACE (ADI) (NON-1100/80) 6-36
6.8.1. Arbitrary Device I/O Packet 6-37
6.8.1.1. Packet Format 6-37
6.8.1.2. Function String Interpretation 6-39
6.8.1.3. Monitor Handling 6-39
6.8.1.4. Disconnecting Channel 6-40
6.8.1.5. Function (EF) Handling 6-40
6.8.1.6. Interrupt Activity Handling 6-40
6.8.2. Initiate ADI and Return Control Immediately (IOARB$) 6-40
6.8.3. Initiate ADI and Exit with Interrupt (IOAXI$) 6-41
6.8.4. Free Format Disk Interface and MSA Tape ADI 6-41
6.8.5. Programming Considerations for Using ADI with Tapes 6-43
6.8.6. I/O Path Selection via the Arbitrary Device Interface 6-44
6.8.7. Auxiliary Storage Interface via the Arbitrary Device Interface 6-45

6.9. ARBITRARY DEVICE INTERFACE (1100/80) 6-46
6.9.1. Arbitrary Device I/O Packet Format 6-46
6.9.2. Device Status Buffer (DSB) 6-53
6.9.3. General Information and Restrictions 6-54
6.9.4. Initiate ADI to Wait for Unsolicited Interrupt (Function = 2) 6-55

6.10. STATUS CODES 6-55

7. File Control 7-1

7.1. INTRODUCTION 7-1

7.2. FILE ORGANIZATION 7-1
7.2.1. Master File Directory 7-1
7.2.2. Mass Storage Allocation 7-2
7.2.3. File Addressing 7-3
7.2.4. Exclusive Use of Files 7-3
7.2.5. Rollout and Rollback of Files 7-3
7.2.6. Retrieving Facility Assignment (FITEM$) 7-4
7.2.6.1. Unit Record and Nonstandard Peripherals 7-6
7.2.6.2. Sector-Formatted Mass Storage Files 7-7
7.2.6.3. Magnetic Tape Peripherals 7-8
7.2.6.4. Word-Addressable Mass Storage Files 7-9
7.2.6.5. Communications Peripherals 7-11
7.2.6.6. Removable Disk Files 7-15
7.2.7. Alternate Methods of Retrieving Facility Assignment Synopsis 7-16
7.2.8. Tape File Initialization (TINTL$) 7-16
7.2.9. Tape Swapping (TSWAP$) 7-17

7.3. TAPE LABELING 7-18
7.3.1. Definitions 7-18
7.3.2. Structure of Magnetic Tape File 7-20
7.3.3. Reading and Writing Tape Label Blocks (TLBL$) 7-20
7.3.4. Reading Tape Label Blocks (LABEL$) 7-26
7.3.5. Multivolume Processing 7-27
7.3.6. Reverse Processing 7-28

7.4. DISK LABELING 7-28

8. Demand Processing 8-1

8.1. INTRODUCTION 8-1
8.1.1. General Demand Terminal Operational Procedures 8-2
8.1.1.1. Initialization 8-2
8.1.1.2. Demand Terminal Modes of Operation 8-3
8.1.1.3. Demand Symbiont Interface 8-3
8.1.1.3.1. Demand Symbiont Control Statements 8-3
8.1.1.3.2. Transparent Control Statements 8-6
8.1.1.3.3. Demand Terminal Messages 8-6
8.1.1.4. Demand Terminal Termination 8-9

8.2. DEMAND SYMBIONTS 8-9
8.2.1. Teletypewriter/DCT 524/500/475 Symbiont 8-9
8.2.1.1. Operational Considerations 8-9
8.2.1.2. Paper Tape Operations 8-10
8.2.1.2.1. Paper Tape Output Operations 8-10
8.2.1.2.2. Paper Tape Input 8-10
8.2.1.3. Special Characters 8-11
8.2.1.4. Interrupting Output Processing 8-11
8.2.1.5. Operation Modification Control Statements (@@TTY,@@DCT) 8-12
8.2.1.6. DCT 524/500/475 in Teletypewriter Mode 8-12

8.2.1.7. Tektronix 4013 8-13
8.2.2. UNISCOPE 100/200/DCT 1000 Symbiont 8-13
8.2.2.1. Operational Considerations for UNISCOPE 100/200 Display Terminals 8-13
8.2.2.2. Operational Considerations for the DCT 1000 8-16
8.2.2.3. Operator Screen and Input/Output Control Statements 8-17
8.2.2.4. Operational Considerations for the UTS 400 8-21

8.3. TERMINAL SECURITY SYSTEM (TSS) 8-21
8.3.1. General 8-21
8.3.2. Demand Mode Logon Modes 8-22
8.3.2.1. Basic Mode 8-22
8.3.2.2. Batch Run From a Demand Terminal 8-23
8.3.2.3. Run Mode 8-23
8.3.2.4. Execution Mode 8-24
8.3.2.5. System Contingencies 8-25
8.3.2.6. Password Modification (@@PASSWD) 8-25
8.3.2.7. Resolicitation of User-id and Password 8-25
8.3.3. Use of TSS Processor 8-26
8.3.4. Error Messages 8-26

8.4. CONSOLE CAPABILITY 8-27

8.5. TERMINAL USER TECHNIQUES 8-32

8.6. EXAMPLE OF A DEMAND RUN 8-34

8.7. USER-SUPPLIED DEMAND SYMBIONTS (RSI$) 8-36
8.7.1. ER RSI$ Functions 8-40
8.7.1.1. Initialization 8-40
8.7.1.2. Input Function 8-41
8.7.1.3. Output Functions 8-41
8.7.1.4. Termination Function 8-42
8.7.1.5. Debugging Functions 8-42
8.7.1.6. General Functions 8-43

9. Communications Handler 9-1

9.1. INTRODUCTION 9-1
9.1.1. Equipment 9-1
9.1.1.1. Communications Terminal Module Controller (CTMC) 9-1
9.1.1.2. C/SP 9-2
9.1.1.3. General Communications Subsystem (GCS) 9-2
9.1.2. Modes of Operation 9-3
9.1.3. Distributed Communications Processor (DCP Series) 9-3

9.2. ASSIGNING LINE TERMINAL (LT) DEVICES 9-4

9.3. THE LINE TERMINAL TABLE 9-4

9.4. COMMUNICATIONS HANDLER OPERATIONS 9-15
9.4.1. Support Operations 9-15
9.4.1.1. Initialization (CMS$) 9-15
9.4.1.2. Dialing (CMD$) 9-16
9.4.1.3. Input (CMI$) 9-17
9.4.1.4. Output (CMO$) 9-18

9.4.1.5. Send and Acknowledge (CMSA$) 9-18
9.4.1.6. Single Buffer Mode for Input/Output Operations 9-18
9.4.1.7. Pool Mode for I/O Operations 9-19
9.4.1.8. Dual Pool Mode for Input Operations 9-21
9.4.1.9. Hangup (CMH$) 9-21
9.4.1.10. Termination (CMT$) 9-21
9.4.2. Communications Pools and Interrupt Tabling Area 9-22
9.4.2.1. Establishing a Communications Pool or Interrupt Tabling Area (CPOOL$) 9-26
9.4.2.1.1. Establishing a Communications Pool 9-26
9.4.2.1.2. Establishing an Interrupt Tabling Area 9-28
9.4.2.2. Removing Buffers from a Pool (CGET$) 9-29
9.4.2.3. Returning Buffers to a Pool (CADD$) 9-29
9.4.2.4. Expanding a Pool (CJOIN$) 9-30
9.4.2.5. Releasing Communications Pool(s) and Interrupt Tabling Area(s) (CREL$) 9-30
9.4.3. Altering Communications Paths (ROUTE$) 9-31
9.4.3.1. Routing Procedures 9-32

9.5. COMPLETION ACTIVITIES 9-32

9.6. IDLE LINE CONTROL 9-33
9.6.1. Idle Line Monitor 9-33
9.6.2. Idle Line Polling 9-34

9.7. TIMING CONSIDERATIONS 9-34
9.7.1. Interrupt Response 9-34
9.7.2. Buffer Processing 9-35

9.8. INFORMATION ANALYSIS 9-36

9.9. ERROR CODES FOR LT CONTINGENCIES 9-36

9.10. PARITY ERRORS 9-37

10. Real-Time Processing 10-1

10.1. INTRODUCTION 10-1

10.2. PROGRAM LOCATION 10-1

10.3. BUFFER OPERATIONS 10-2
10.3.1. Transmission Types 10-2
10.3.2. Main Storage Availability 10-2
10.3.3. Pool Size 10-3
10.3.4. Buffer Size 10-3
10.3.5. Dual Pool Method 10-4

10.4. PROGRAM EXECUTION CONSIDERATIONS 10-5
10.4.1. Priority Categories 10-5
10.4.1.1. I/O Priority 10-5
10.4.1.2. Dispatching Priority 10-5
10.4.2. Priority Control 10-7
10.4.2.1. Changing Activity Priorities (RT$ and NRT$) 10-7
10.4.2.2. Application of Multiprogramming to Real-Time 10-7
10.4.2.3. Interrupt Activity Priority Reduction (UNLCK$) 10-8
10.4.2.4. Activity Termination (EXIT$) 10-8

10.4.2.5. Timed Wait Considerations 10-8
10.4.2.6. Console Interrupt Handling 10-9
10.4.3. Test and Set Usage 10-9

10.5. PROGRAMMER'S GENERAL RESPONSIBILITIES 10-10

10.6. ESI CONSIDERATIONS 10-11
10.6.1. ESI Activity Concept 10-11
10.6.2. ESI Timing 10-11
10.6.2.1. ESI Interrupts 10-11
10.6.2.2. Real-Time Activities 10-12

11. Checkpoint/Restart 11-1

11.1. INTRODUCTION 11-1
11.1.1. Uses of Checkpoint/Restart 11-1
11.1.1.1. Insurance in Case of System Failure 11-1
11.1.1.2. Spreading Executions of Long Jobs over Several Computing Sessions 11-1
11.1.1.3. Insurance In Case of Program Failure 11-1
11.1.2. User Interface 11-1

11.2. CHECKPOINT 11-1
11.2.1. Control Statement (@CKPT) 11-2
11.2.2. Executive Request 11-2

11.3. RESTART 11-2
11.3.1. Control Statement (@RSTRT) 11-2
11.3.2. Executive Request 11-3
11.3.3. Initializing Restarts 11-3
11.3.4. Restart Contingencies 11-4
11.3.5. File Handling at Restart 11-6

11.4. ERROR MESSAGES AND ERROR CODES 11-7

12. Internal Executive Design 12-1

12.1. INTRODUCTION 12-1

12.2. BASIC DESIGN PHILOSOPHY 12-1

12.3. EXECUTIVE MAIN STORAGE USAGE 12-3
12.3.1. General Layout and Discussion 12-3
12.3.1.1. 1106, 1108, 1100/10/20 Main Storage 12-3
12.3.1.2. 1110, 1100/40 Main Storage 12-5
12.3.1.3. 1100/80 Main Storage 12-6
12.3.1.4. Executive Addressing Windows 12-7
12.3.2. PCT Usage 12-8
12.3.3. Definition and Residency of Components 12-8

12.4. MULTIPROCESSING 12-9

12.5. SCHEDULING 12-10
12.5.1. General 12-10
12.5.2. Facilities Inventory and Selection 12-10
12.5.3. Control Statement Interpreter (CSI) 12-12

12.5.4. Coarse Scheduler 12-12
12.5.5. Dynamic Allocator 12-15
12.5.5.1. General Overview 12-15
12.5.5.2. Dynamic Main Storage Allocation 12-16
12.5.5.3. Demand/Batch Sharing 12-18
12.5.5.4. Time Sharing 12-18
12.5.6. Dispatcher 12-19
12.5.6.1. Interlock Processing 12-19
12.5.6.2. Switching 12-20

12.6. CLOCKING 12-21
12.6.1. Real-Time Clock 12-22
12.6.2. Day Clock 12-22
12.6.3. Quantum Timer 12-22

12.7. INTERRUPT HANDLING 12-22
12.7.1. Input/Output Interrupts and Queuing 12-22
12.7.2. Interprocessor Interrupts 12-23
12.7.3. Hardware Fault Interrupts 12-23
12.7.3.1. Storage and Control Register Parity Error Interrupts 12-23
12.7.3.2. Power Loss Interrupts (1106/1108 only) 12-24
12.7.4. Program-Generated Interrupts 12-24

12.8. CATALOGED FILE RECOVERY 12-25

12.9. REMOVABLE DISK RECOVERY/REGISTRATION 12-26

Appendix A. EXEC Control Statements A-1

A.1. SUMMARY OF CONTROL STATEMENTS A-1

A.2. SUMMARY OF DEMAND SYMBIONT CONTROL STATEMENTS A-8

Appendix B. Summary of Executive Requests B-1

Appendix C. Diagnostic Messages and Status Codes C-1

C.1. RUNSTREAM DIAGNOSTIC MESSAGES C-1
C.1.1. General C-1
C.1.2. Diagnostic Messages C-2

C.2. FACILITY REQUEST STATUS CODES C-16

C.3. ERR MODE (EMODE) AND I/O STATUS CODES C-18

C.4. CSF$ EXECUTIVE REQUEST STATUS CODES C-35
C.4.1. Facility Request Status Codes (@CAT, @ASG, @FREE, @MODE, @USE) C-35
C.4.2. @SYM and @BRKPT Status Codes C-35
C.4.3. @ADD Status Codes C-36
C.4.4. @START Diagnostics and Status Codes C-36

C.5. GUARD MODE AND UNDEFINED SEQ. STATUS (1110, 1100/40) C-37

C.6. GUARD MODE AND ADDRESSING EXCEPTION STATUS FOR 1100/80 C-38

Appendix D. Conversion Tables D-1

D.1. INTRODUCTION D-1

D.2. ASCII AND FIELDATA CONVERSION TABLES D-1

D.3. SPECIAL CHARACTERS IN ASCII D-4

D.4. UNISCOPE 100/200, UTS 400 DISPLAY TERMINAL D-5

D.5. BINARY/HEXADECIMAL CONVERSION TABLE D-7

D.6. OCTAL/DECIMAL CONVERSION TABLE D-7

D.7. 80-COLUMN CARD CODE CONVERSION TABLE D-12

D.8. EBCDIC/ASCII 80-COLUMN CARD CODE CONVERSION TABLE D-14

Appendix E. Equipment Codes E-1

Appendix F. Run Setup Examples F-1

F.1. ASSEMBLER EXAMPLE F-1

F.2. LEGENDRE POLYNOMIAL GENERATION F-2

F.3. ASSEMBLER PROGRAM UPDATE EXAMPLE F-4

F.4. FORTRAN PROGRAM EXAMPLE F-5

F.5. TAPE FILE MANIPULATION F-8

F.6. TAPE FILE BUILD EXAMPLE F-9

F.7. UNISCOPE TERMINAL BREAKPOINT TO HIGH SPEED PRINTER F-12

F.8. START RUN EXAMPLE F-16

User Comment Sheet

Index

Figures

Figure 6-1. I/O Packet, Mass Storage, and Magnetic Tape Peripheral 6-2
Figure 6-2. I/O Packet for EACQ$ Function 6-30
Figure 6-3. Arbitrary Device Packet 6-37
Figure 6-4. Arbitrary Device Handler Request Packet 6-46
Figure 6-5. User-Specified Path Format 6-48
Figure 12-1. 1106, 1108, 1100/10/20 Main Storage 12-3
Figure 12-2. 1110, 1100/40 Main Storage 12-5
Figure 12-3. 1100/80 Main Storage 12-6

Tables

Table 3-1. Summary of Executive Control Statements 3-5
Table 3-2. @RUN Control Statement, Options 3-8
Table 3-3. @MSG Control Statement, Options 3-24
Table 3-4. Sector-Formatted Mass Storage @ASG Control Statement, Options 3-38
Table 3-5. Magnetic Tape @ASG Control Statement, Options 3-46
Table 3-6. @CAT Control Statement, Options 3-57
Table 3-7. @FREE Control Statement, Options 3-59
Table 3-8. Processors that Use the SI, SO, and RO Parameters 3-67
Table 3-9. Processors that Require the SI and SO Parameters 3-68
Table 4-1. Available ERs 4-3
Table 4-2. Common Bank Reload Type 4-44
Table 4-3. Contingency Types 4-55
Table 4-4. Error Types 4-57
Table 4-5. EXEC Codes Currently Assigned 4-82
Table 5-1. Bit Settings Returned in Control Register A0 for READ$ Request 5-4
Table 5-2. Print Control Functions 5-15
Table 5-3. Element Format for Cartridge ID 5-18
Table 5-4. Punch Control Functions 5-21
Table 5-5. SYMB$ Packet 5-28
Table 6-1. Octal and Mnemonic I/O Codes Defined in SYS$*RLIB$ 6-4
Table 6-2. Magnetic Tape I/O Functions and Codes 6-14
Table 6-3. Byte Channel Data Word Formats 6-22
Table 6-4. Magnetic Tape Function versus Unit Type 6-25
Table 6-5. Word-Addressable Mass Storage I/O Functions and Codes 6-27
Table 6-6. Mass Storage I/O Functions and Codes 6-32
Table 6-7. Device Status Buffer 6-53
Table 7-1. Equipment Codes Returned by FITEM$ 7-13
Table 7-2. Read/Write System Label Packet 7-21
Table 7-3. Read/Write User Label Packet 7-24
Table 7-4. Error Codes and their Meanings 7-25
Table 8-1. Demand Symbiont Interface Control Statements 8-4
Table 8-2. Keyins and Console Modes Allowed 8-30
Table 8-3. RSI$ Output Functions 8-41
Table 9-1. LTT Output-Status Codes 9-6
Table 9-2. LTT Input-Status Codes 9-9
Table 11-1. Cataloged File Handling 11-6
Table 11-2. Cataloged Mass Storage Reloading 11-6
Table 11-3. Checkpoint/Restart Error Codes 11-7
Table 11-4. Checkpoint/Restart Messages 11-9
Table 12-1. Resident Components of the Executive System in Main Storage 12-8
Table 12-2. Nonresident (Transient) Components of the Executive System 12-9
Table C-1. Facility Status Bits C-16
Table C-2. Error Mode (EMODE) and I/O Status Codes C-19
Table C-3. @SYM and @BRKPT Status Codes C-35
Table C-4. @START Status Code C-37
Table D-1. Fieldata-to-ASCII Conversion D-2
Table D-2. ASCII-to-Fieldata Conversion D-3
Table D-3. Cursor/SOE Coordinates D-6
Table D-4. Binary/Hexadecimal Conversion D-7
Table D-5. Octal/Decimal Conversion D-8
Table D-6. Fieldata 80-Column Card Code Conversion Table D-12
Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion D-14

1. Introduction

1.1. SCOPE OF MANUAL

The SPERRY UNIVAC Series 1100 Operating System consists of Sperry Univac–supplied software for the SPERRY UNIVAC Series 1100 Systems. This volume and Volumes 3 and 4 discuss the base portion of the operating system, that is, the SPERRY UNIVAC Series 1100 Executive System (EXEC 8) and the associated software needed to construct, execute, and maintain user programs. Information on language processors such as COBOL, FORTRAN, and the Assembler, and on applications software such as SORT/MERGE, APT, and OPTIMA can be found in their respective manuals.

Information that is primarily of interest only to an operator, installation manager, or system analyst is described only briefly if at all (e.g., operating procedures, system generation procedures, internal system logic, etc.). Such material is covered in other Sperry Univac publications.

The purpose of this manual is to provide information so the user programmer can make full use of the wide range of capabilities provided by the SPERRY UNIVAC Series 1100 Executive. Any differences between the operating system described in this manual and the latest released software are described in the Software Release Documentation that accompanies each release.

A knowledge of the Series 1100 hardware architecture and machine (assembler) language programming is assumed. This knowledge is helpful, but not mandatory, for the user of a higher level language or applications package.

1.2. THE OPERATING SYSTEM

The SPERRY UNIVAC Series 1100 Operating System is designed to meet the total computing requirements of today's users, and to allow for the change and growth required for the future. The operating system is the outgrowth of Sperry Univac's many years of experience in multiprogramming, multiprocessing, time sharing, communications, and real–time oriented systems. It provides complex environments and, yet, is easy to operate and use.

A complete set of software, ranging from high-level language compilers to basic service functions, is included in the operating system. The six major categories are:

- Executive System (EXEC 8)
- System Processors
- System Utility Routines
- Language Processors
- Subroutine Library
- Applications Programs

The first three categories are discussed in detail in this volume, and Volumes 3 and 4. The execution environment is specified for the software falling into the last three categories. In addition to the standard operating system, this manual describes certain utility routines not required for the site to be operational, but provided only for the convenience of the user. These utility routines receive a lower level of support and are as follows:

- Document Processor (DOC) – see Volume 4
- Element Listing Routine (LIST) – see Volume 4
- Cross-reference Processor (CULL) – see Volume 4
- Program Trace Routine (SNOOPY) – see Volume 4

1.3. THE EXECUTIVE SYSTEM (EXEC 8)

To take full advantage of the speed and hardware capabilities of the Series 1100 Systems and to make effective use of a given hardware configuration, a comprehensive internal operating environment has been created.

This environment permits the concurrent operation of many programs; it allows the system to react immediately to the inquiries, requests, and demands of many different users at local and remote stations; it accords with the stringent demands of real-time applications; it can store, file, retrieve, and protect large blocks of data; and it makes the best use of all available hardware facilities, while minimizing job turnaround time.

Only through central control of all activities of the system can this environment of combined hardware and software systems be fully established and maintained to satisfy the requirements of all applications. The responsibility for efficient, flexible, centralized control is borne by the Executive System, which controls and coordinates the functions of the complex internal environment. By presenting a relatively simple interface, it allows the programmer to use the system easily, while relieving concern for the internal interaction between the program and other coexistent programs.

1.3.1. Multiple Modes of Operation

The technical capabilities of the Executive System cover a great variety of data processing activities. Its design offers versatility to handle batch processing, demand processing and real-time processing, using multiprogramming and multiprocessing techniques. An installation that is not interested in making use of the full range of capabilities may specify component features to be eliminated at system generation.

1.3.1.1. Batch Processing

Among the capabilities of the Executive System is the support provided for batch processing. The system is designed to ease run preparation and submission, to shorten job turn-around time, and to reduce the need for operator intervention and decisions. Batch jobs may be processed from a variety of remote terminals, as well as from central site equipment.

1.3.1.2. Demand Processing (Time Sharing)

The batch processing capabilities of the Executive System include its time sharing capabilities. This mode of operation accommodates simultaneous requests and demands from users at numerous remote inquiry terminals operating in a demand (or conversational) mode. All facilities available to the batch processing user are also available in the demand mode, the primary difference being that the Executive System permits the user additional flexibility in the statement and control of individual runs. For example, when an error is made the user simply corrects it online and proceeds rather than suffering the turn-around cycle inherent in batch processing. The demand user may interact with the system at various levels which include direct communications with either the Executive System or a user program or with a conversational processor, such as BASIC.

1.3.1.3. Real-Time Processing

The Executive System is also designed to be applicable to programs which have real-time requirements. The UNIVAC Communications Subsystem, together with efficient scheduling and interrupt processing features of the Executive System, provides an environment for the operation of real-time programs.

1.3.1.4. Transaction Processing Interface

Another type of remote inquiry terminal processing provided by the Executive system is transaction processing. Selected programs may be executed on request from remote inquiry terminals. The types of operations performed in this processing mode are more limited than in the demand processing mode. However, the number of remote terminals and the number of inquiries per second is much greater. The types of operations allowed are prespecified to the Executive System. They may be supplied by Sperry Univac or they may be specially written.

1.3.1.5. Multiprogramming and Multiprocessing

Runs may come from many sources, remote and central. These various runs, through the Executive System's use and control of efficient multiprogramming and multiprocessing techniques may, at any given moment, be in different stages of activity. Input, processing, and output may all be occurring simultaneously, thus allowing efficient operation.

1.3.2. Utilization of Mass Storage

The Executive System is designed to ensure effective and efficient utilization of the mass storage devices. The consequence is an unprecedented ability to relieve operators and programmers of the responsibility of maintaining and handling cards and magnetic tapes, thus eliminating many of the errors which heretofore have accompanied the use of large scale software systems. At the same time, the overall operating efficiency is considerably improved.

Permanent data files and program files are maintained on the mass storage devices, with full facilities for modification and manipulation of these files. Security measures are established by the Executive System to ensure that files are not subject to unauthorized use. Provisions are also made within the Executive System for automatic relocation of infrequently used files to magnetic tape, as unused mass storage space approaches exhaustion. When the use of files relocated in such a manner is requested, they are retrieved and restored under control of the Executive System without any need of intervention by the user.

1.3.3. Functional Areas of the Executive System

1.3.3.1. Executive Control Language

In the Executive System, the user has a simple means of directing the execution of the individual tasks of a run and of relaying to the Executive operational information concerning the run. This is accomplished through a set of control statements capable of performing all of the functions desirable and necessary in a modern executive system. The control language is open ended and easily expanded, so that features and functions may be added as needed.

The basic format of a control statement is quite simple, and is adaptable to a large number of input devices. Each control statement consists of a heading character @ (master space or commercial at symbol), or in special cases, an @@ for recognition purposes, followed by a command and a variable number of parameters. The end of a control statement is indicated by the end of a card, a carriage return, or an equivalent signal, depending on the type of input device.

1.3.3.2. The Supervisor

The supervisor is the Executive System component that controls the sequencing, setup, and execution of all runs. It is designed to control the execution of a large number of independent and interdependent programs.

The supervisor contains three levels of scheduling: coarse scheduling, dynamic allocation of storage space, and central processor unit (CPU) dispatching. Runs entering the system are sorted into information files and these files are used by the supervisor for run scheduling and processing. Control statements for each run are retrieved and scanned by a control statement interpreter in the supervisor to facilitate the selection of runs for setup by the coarse scheduler. The coarse scheduling of each run primarily depends on two factors: the priority of the run, and its facility requirements.

The dynamic allocator takes runs set up by the coarse scheduler and allots main storage space according to the needs of the individual tasks (programs) of each run. Normally, tasks from many different runs are located in main storage at the same time. Each run may be thought of as being made up of tasks, where a task is a single operation of a system processor or the execution of a user program. All tasks for a given run are processed serially but not necessarily consecutively. If there are several runs, the tasks of separate runs are interleaved.

When time sharing of main storage is appropriate (e.g., the total main storage requirements of all programs in a state ready for execution are greater than the available main storage), the dynamic allocator initiates storage swaps. This involves writing one or more programs from main storage to mass storage and allocating the now available main storage to other programs. Such action is taken to provide reasonable response time to remote demand processing terminals, or to satisfy batch priority requirements.

The CPU dispatching routine is the third level of scheduling. It selects among the various tasks whenever it is appropriate to switch the commitment of the CPU from one task to another. Under

normal circumstances, a batch program is allowed to use a CPU either until it becomes interlocked against some event or until some higher priority program is freed of all of its interlocks. On multiprocessor systems, two or more tasks will be in actual execution at the same time.

1.3.3.3. Facilities Assignment

Available facilities and their dispositions are indicated to the system at system generation; thereafter, the Executive System assigns these facilities, as needed and as available, to fulfill the facilities requirements of all runs entering the system. The Executive System maintains current inventory tables that indicate what facilities are available for assignment, and which runs are using the currently assigned facilities.

1.3.3.4. File Control

The Executive file control routines afford the highest degree of operational flexibility in storing and retrieving data, without concern for the physical characteristics of the recording devices. Thus, most files are made insensitive to input/output (I/O) media characteristics, as the system adjusts the interface between the file and the device. Security measures ensure that files are not subject to unauthorized use or destruction. File control routines are provided to roll out files from mass storage devices to magnetic tape, as well as to reconstruct such files on the mass storage devices when the user calls for them.

Comprehensive utility routines are available for manipulation of files and for informing the user of current status and structure of files. Provisions are made for random storage and retrieval of data, under the direction of the user. User program files and data files are maintained and processed in the same environment.

1.3.3.5. Operator Communications

Operator functions are required for a large variety of activities. The Executive System groups them into classes, thus allowing the division of operator duties in a multiprocessor installation. These functions may be associated with as many as three (1106, 1108, 1100/10/20) or four (1110, 1100/40) or more (1100/80) system consoles or as few as one, depending on the complexity and layout of the installation.

The Executive System displays information such as current system load and operator requests associated with I/O setup and I/O interlocks. The operator can request other information, such as backlog status. If the display area becomes filled, the Executive defers lower priority displays.

Since this manual is for the user programmer as opposed to the computer operator, it does not contain detailed information concerning the operator communications functions.

1.3.3.6. Input/Output Device Handlers and Symbionts

The input/output device handlers and symbionts control the activities of all I/O channels and peripheral equipment attached to the system.

The following is a list of the Sperry Univac onsite and remote peripheral hardware that is supported by the Executive:

- Mass Storage Devices
  - FH-432 and FH-1782 Magnetic Drums
  - Unitized Channel Storage (1106, 1108, 1100/10/20)
  - 8405 Disk
  - 8414 Disk
  - 8424 Disk
  - 8425 Disk
  - 8430 Disk
  - 8433 Disk
  - 8434 Disk
  - 8440 Disk
  - 8450 Disk
  - 8460 Disk
  - FASTRAND II and III Magnetic Drums
- Magnetic Tape Devices
  - UNISERVO 12, 14, 16, 20, 30, 32, 34, and 36 Magnetic Tape Units
  - UNISERVO VIC and VIIIC Magnetic Tape Units (except 1100/80)
- Onsite Symbiont Devices

  Printer Devices
  - Type 0751, 0755, and 0758 High Speed Printers (except 1100/80)
  - Type 0768, 0770, and 0776 High Speed Printers
  - UNIVAC 9200/9300 Series Printer (except 1100/80)
  - UNIVAC 1004 Printer (except 1100/80)

  Card Devices
  - Type 0706 and 0716 Card Readers (only 0716 on 1100/80)
  - Type 0600 and 0604 Card Punches (only 0604 on 1100/80)

- UNIVAC 1004 Card Punch and Card Reader (except 1100/80)
- UNIVAC 9200/9300 Series Card Readers and Card Punches (except 1100/80)

■ Communications Devices

- Distributed Communications Processor (DCP)
- Onsite Interface Hardware

  Communications/Symbiont Processor (C/SP)
  Communications Terminal Module Controller (CTMC)
  General Communications Subsystem (GCS)

- Remote Terminal Hardware

  UNISCOPE 100/200 Display Terminal
  Universal Terminal System (UTS 400, UTS 700)
  DCT 2000, 1000, 524, 500, and 475 Data Communications Terminals
  9300/9300-II Remote System
  9200/9200-II Remote System
  1004 Card Processor
  90/30 Remote System

NOTE:

In addition to the above, the TEKTRONIX Model 4013, and TELETYPE Models 33, 35, 37, and 38 are supported. The Tektronix Model 4013 and Teletype Model 38 are supported only via the CTMC and GCS.

1.4. SYSTEM PROCESSORS

The system processors (see Volume 3) of the operating system are programs which provide for the utilitarian functions required to construct and modify programs, maintain and modify files, and provide diagnostic information upon program termination.

1.4.1. Collector (MAP)

The Collector is designed to provide the user with the means of collecting and linking relocatable subprograms to produce an absolute program in a form ready for execution under control of the Executive System.

1.4.2. File Utility Processor (FURPUR)

FURPUR consists of a set of file maintenance routines which provide the flexibility in management and manipulation of cataloged or temporary files containing data or programs.

1.4.3. Postmortem Dump Processor (PMD)

The postmortem dump processor (PMD) produces edited dumps of the contents of main storage at program termination; dumps produced dynamically during execution are automatically printed. Individual program parts are identified with the assistance of diagnostic tables produced with the absolute program by the Collector.

1.4.4. DATA Processor

The DATA processor is used to introduce, update, and correct data files from the control stream.

1.4.5. ELT Processor

The ELT processor is used to introduce an element into a particular program file or make corrections to a symbolic element in a program file from the runstream.

1.4.6. File Administration Processor (SECURE)

The SECURE processor uses a source language structure which allows the user to define specific tasks with simple COBOL–like statements. This processor's primary functions are to produce backup copies of cataloged files, and to provide a recovery mechanism for these files in case of system failure.

1.4.7. Text Editor (ED)

The ED processor is a text editor which enables a user to modify or move character strings in either program files or data files.

1.4.8. Procedure Definition Processor (PDP)

The procedure definition processor (PDP) accepts source language statements defining assembler, COBOL, or FORTRAN procedures and builds an element in the user–defined program file. These procedures may be referenced subsequently in an assembly or compilation without definition.

1.4.9. SSG Processor

The SSG processor is a general–purpose symbolic stream generator. Any variety of symbolic streams, varying from a file of data to a runstream which configures an executive system, may be generated. Directions and models for building of the desired stream images are conveyed to SSG through a skeleton which is written in SYMSTREAM, an extensive manipulation language.

1.4.10. Conversational Time Sharing (CTS)

The Conversational Time Sharing (CTS) system is an interactive interface to users of SPERRY UNIVAC Series 1100 Systems. CTS provides a complete set of system control commands designed to be easy to learn and simple to use without knowledge of the existing SPERRY UNIVAC Series 1100 Executive. CTS commands are conversational and provide an information retrieval system which contains information about every command in CTS. CTS also includes a sophisticated programmable editor with language syntax analysis capabilities.

1.4.11. High Volume Time Sharing (HVTS)

The High Volume Time Sharing (HVTS) System provides users access to the Sperry Univac Series 1100 System. The HVTS commands used to enter, edit, and execute programs allow the omission of parameters. Defaults are automatically provided by HVTS for the missing parameters. Using HVTS, system security is provided by user-ids and passwords which are required when a user is logging onto the system, and by the different levels of privacy allowed for accessing other users' files.

1.5. SYSTEM UTILITY ROUTINES

The system utility routines (see Volume 4) provide features which are commonly used. Unlike the system processors, the features provided are not necessary for the effective utilization of the operating system.

1.5.1. CULL Processor

The CULL processor produces an alphabetically-sorted, cross-referenced listing of all symbols in a specified set of symbolic elements. Provisions are included, via options, to selectively include or exclude defined symbols or symbol groups from the output.

1.5.2. Document Processor (DOC)

The DOC processor composes file input into document format according to the user's specifications. Control statements provide listing and text control, including pagination, justification, indentation, and hyphenation.

1.5.3. LIST Processor

This special-purpose processor provides edited element listings which include associated element control information not normally of interest to the user. It is intended for debugging of software which deals with program files.

1.5.4. 1100 Virtual Machine Control Processor (FLIT)

FLIT provides a virtual system environment for the Series 1100 user. The virtual system is realized through simulation of the requested processor, memory, and peripherals. Any Series 1100 system may be simulated on any host system. Two main modes of simulation are provided. In system mode, the basic hardware is simulated so that FLIT executes control programs such as the Series 1100 Executive. In user mode, the normal user program environment is simulated. In this mode, users may execute individual programs just as these programs would be executed under 1100 Executive control. In both modes, FLIT provides extensive facilities for analysis, control, and modification of the executing program.

1.6. LANGUAGE PROCESSORS

The operating system provides many language processors, such as FORTRAN, COBOL, NUALGOL, PL/1, and the Assembler. Certain of these processors are specifically designed for demand mode operation. Consult the SPERRY UNIVAC Series 1100 Summary of Current Documentation, UP-7893, for information on using a particular language.

1.7. RELOCATABLE SUBROUTINE LIBRARY

An extensive library of relocatable subroutines is provided by the operating system. Subroutines referenced by user programs are automatically included when the absolute program is constructed by the Collector. The library elements included fall into the following general categories:

- Subroutines that support higher level languages (COBOL Library, FORTRAN Library, etc.)
- Processor Interface Routines
- SORT/MERGE
- Diagnostic Subroutines
- Service Routines, for editing, conversion, segment loading, etc.
- MATH-PACK and STAT-PACK Mathematical and Statistical Functions
- Assembler Procedure Library - Provides macro capability for generation of common machine-level coding and parameter sequences.

This volume and Volumes 3 and 4 describe only those subroutines that fall into the base portion of the operating system, such as the diagnostic subroutines, the editing routines and certain assembler procedures. Consult the appropriate manual for information on a subject not covered in this document.

1.8. APPLICATIONS PROGRAMS

The operating system provides many applications programs such as APT, DMS, FMPS, GPSS, and OPTIMA. Consult the SPERRY UNIVAC Series 1100 Summary of Current Documentation, UP-7893, for information on using a particular application program.

SPERRY UNIVAC

PUBLICATIONS RELEASE

Series 1100

Executive System
Volume 2
EXEC Level 36R2

Programmer Reference

UP-4144.23

This Sperry Univac Series 1100 Library Memo announces the release and availability of "SPERRY UNIVAC Series 1100 Executive System, Volume 2, EXEC Level 36R2, Programmer Reference," UP-4144.23. This is a Standard Library Item (SLI).

This manual provides information on enhancements in level 36R2 of the Executive System in the following areas:

- Integrated Recovery, Audit Trail, and Rollback
- Quota; Mass Storage Utilization and Units Assigned Enforcements
- MSCON DMBT$, DMBTS$, and MUSER$ Subfunctions
- Reduction of I/O to GENF$
- 8450 (Movable and Fixed Head) Disk
- Wait for Unsolicited Interrupt

This manual corresponds to level 36R2 of the Series 1100 Executive. The Software Release Documentation accompanying each new level of the Executive will contain the information needed to modify the current UP documentation. The Series 1100 Executive level 36R2 software described in this document is a proprietary product of Sperry Univac. As such, authorization to receive the Executive level 36R2 software must be established by contractual agreement prior to transmittal.

Additional copies of this manual may be requisitioned by your Sperry Univac representative.

Users of EXEC level 36R1 and earlier levels should continue to use UP-4144.2, UP-4144.21 plus appropriate update packages, UP-4144.22, or PUP-4144.2P2 until they have moved up to EXEC level 36R2.

For users of EXEC level 36R2, UP-4144.23 supersedes and replaces "SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Level 35R1, Programmer Reference," UP-4144.22 released by Library Memo dated December 1978. Also superseded and replaced are UP-4144.2, UP-4144.21, Update Packages A, B, C, and D for UP-4144.21, UP-4144.22, and PUP-4144.2P2. These are the Volume 2 manuals and update packages for EXEC levels 32R1, 33R1, 33R2, 33R3, 35R1, and 36R1.

To Mailing Lists AC, BZ, CZ, and MZ.	Library Memo plus UP-4144.23 (588 pages and cover) to Mailing Lists 8, 9, 37, 38, 62, 63, 64, 81, 83, and 89.	Library Memo for UP-4144.23

July 1979

2. General Concepts and Definitions

2.1. INTRODUCTION

This section presents certain basic information that is essential for comprehension of the remaining sections of the manual which deal with specific areas of the operating system.

2.2. DEFINITIONS AND ABBREVIATIONS

The following paragraphs (2.2.1 and 2.2.2) define terms that aid in comprehending the remainder of the manual. The reader is encouraged to become familiar with them before proceeding.

2.2.1. Definitions

NOTE:

The hardware organization of the 1110, 1100/40, 1100/80 Systems is somewhat different from that of 1106, 1108, 1100/10/20 Systems. In some instances, different terms have been adopted for functionally similar components. In such cases, to avoid confusion and improve readability, the 1108 term has, as a general rule, been used throughout this document synonymously with the corresponding term on other 1100 Systems, except where such usage would be misleading. The principal corresponding terms are:

1106, 1108, 1100/10/20	*1110, 1100/40*	*1100/80*
CPU	*CAU (plus IOAU)*	*CPU (plus IOU)*
ACU	*SPU*	*STU*
ICR	*GRS*	*GRS*

absolute element – An element containing a complete program in binary form suitable for execution by the Executive. Such elements normally occur as output from a collection of relocatable elements, with all necessary linkages and relocation performed.

active PSR – On 1110, 1100/40/80 Systems, that Processor State Register (PSR) (main or utility) under which the current instruction is being executed.

activity – Formally, a virtual CPU. The Executive maintains a CPU environment (current P-register value, control register contents, and so forth) for an execution sequence or thread, called an activity, such that the activity appears to have continuous use of a single CPU as long as it desires, even though the Executive may interleave CPU usage among many activities and execute them on different CPUs. All program execution is by activity. A program is initially assigned and typically needs just one activity; complex programs may register additional activities to be executed asynchronously.

activity-id – A special-purpose numeric identifier which may be acquired upon registration of a new activity. An activity having such an identifier may wait for the termination of one or more other activities having an id. Activity-id is not to be confused with activity name; their functions are separate and independent.

activity-name – A general-purpose identifier acquired by an existing activity to allow other activities of the same program to communicate or synchronize with it.

activity registration – The act of creating and registering a new activity with the Executive. Also known as forking.

activity termination – The permanent cessation of execution by an activity. This is normally done voluntarily by the activity itself, but may also result from an externally-initiated action such as an abort sequence. A program terminates when all of its activities have terminated.

ACU – Availability Control Unit (1106, 1108, 1100/10/20). Partitions the hardware into specific configurations by disabling and enabling the interface between units. The ACU can also take units offline for maintenance without disrupting operation of the remainder of the system.

ACW – Access Control Word. A word defining the length and location of a data area in main storage, most commonly an I/O buffer.

application – The total hardware configuration, or a subset resulting from partitioning that configuration by using either the availability control unit (ACU), the system partitioning unit (SPU), the system transition unit (STU), or software downing of components.

auxiliary storage – Supplemental storage, not directly addressable by a CPU and accessible only via I/O (as opposed to main storage) and typically having much greater capacity. Includes magnetic tape, flying-head magnetic drum, FASTRAND drum, disk, or unitized channel storage.

AXR$ – A system procedure that contains the numeric definitions of the standard mnemonic designators for control registers, partial word designators, etc., which are used in assembly language coding.

bank – The basic program entity for which storage allocation is required. When in main storage, a bank by definition occupies contiguous

physical addresses. In the normal user program sense, a bank is a Collector–defined portion of the program which is specified by a single Bank Descriptor Word (BDW).

active bank – Any bank which must be in main storage for a program to execute. This includes all static banks and all dynamic banks that are in use.

based bank – A bank currently in use by one or more activities, in that the activities have a base register pointing to the bank. A based bank is in use by definition.

common bank – A bank that can be an active part of more than one program at a time. Common banks are generally reentrant.

dynamic bank – A bank which may require Executive intervention whenever entered (or left) via LBJ/LIJ/LDJ. Dynamic banks are loaded only when actually in use (based, or with outstanding I/O).

static bank – A bank which can be entered (or left) via LBJ/LIJ/LDJ without requiring Executive intervention. Static banks are loaded in main storage whenever the programs of which they are a member are loaded.

batch processing – A mode in which runs are processed without any basic requirement for interactive manual data or control input (such as from a keyboard) during processing, as opposed to demand processing.

breakpoint (symbiont) – Division of symbiont–defined files into parts such that the output of completed parts may be initiated prior to run completion. This procedure allows more efficient utilization of printers and punches when large symbiont output files are involved.

cataloged file – A file known to and retained by the Executive, for an indefinite period not necessarily related to the life of a particular run, and generally retrievable by runs other than the run which originally created the file. In some cases, a cataloged file may be accessed simultaneously by two or more runs.

CAU – Command/Arithmetic Unit. The 1110, 1100/40 equivalent of the instruction processing portion of a CPU; input/output is controlled in a separate unit (IOAU) on 1110, 1100/40 Systems.

central site – The CPUs, main storage, and attached onsite peripheral equipment in a particular application.

CERU$ – A system relocatable library element that defines the numeric index associated with the mnemonic designation of each Common Bank Descriptor Index (BDI).

collection – The process by which individual (relocatable) elements are combined to form a complete program (absolute element). This process begins with explicit specification of elements to be included, and typically involves inclusion of additional unspecified elements required to satisfy undefined references

(these are most commonly obtained from the system relocatable subroutine library RLIB$).

Collector – A system processor that provides the collection function.

communications device – An input or output device which operates in a real-time mode. CPU(s) must be prepared to receive input at any time or information may be lost.

contingency – An abnormal or unanticipated event requiring special action, and usually causing diversion of an activity's execution path to a specially prepared routine or to a standard action sequence.

control registers – Those operating registers of a CPU which can be utilized directly by a program; that is, the X-, A-, and R- registers (except X0 and R0).

control statement – A data image used to direct the Executive in processing a run. A control statement is identified by a master space or at symbol (@) in column 1.

core or core storage – In general usage, main storage.

CPU – Central Processing Unit. A unit of the 1106, 1108, 1100/10/20, and 1100/80 Systems containing circuitry and operating registers which control the interpretation and execution of instructions. A CPU does not contain any main or auxiliary storage. Under the Executive, multiple CPUs may access common main and auxiliary storage. The CPU also includes input/output control registers and circuitry in 1106, 1108, and 1100/10/20 systems. (See introductory note on CAU.)

CRT – Cathode Ray Tube (CRT). Used to denote any of several supported remote terminals which incorporate a CRT as the output display device as opposed to a typewriter.

cycle – A number used to differentiate successive updates of files or symbolic elements.

DATA file – A file in SDF created or updated by one of several operating system mechanisms, usually a system processor called DATA. Not to be confused with the generic term "data file".

data image or image – A data record in human intelligible (character) format; most commonly refers to punched card or printer data.

deadline run – A deadline run is a batch run which is afforded certain scheduling priorities to assure run completion by a prespecified time. Except for these scheduling exceptions, deadline runs are treated as batch runs by the Executive.

demand processing – A mode in which run processing is basically dependent on manual interaction with the system during processing, typically from a remote site. Also commonly known as "time-sharing", as opposed to batch processing.

element – A named grouping of information typically manipulated as a unit, and typically defining a logical program part such as a subroutine. There are four basic types of elements: symbolic, relocatable, absolute, and omnibus.

element file – A specially structured file containing a group of elements, residing on magnetic tape, as opposed to a program file. The arbitrary distinction between the two file types is made to avoid confusion between operations that may be done on one medium but not the other.

ER – Executive Request. An instruction which causes a special interrupt used to request Executive service (e.g., I/O, time of day, etc.). Also, the service resulting from the request. This is the standard interface between programs and the Executive.

ERU$ – A system relocatable library element that defines the numeric index associated with the mnemonic designation of each Executive service request (ER).

ESI – Externally Specified Index. A mode of I/O operation for a SPERRY UNIVAC Series 1100 computer I/O channel through which many communications peripheral devices are multiplexed into one I/O channel. Each communications line has its own area in main storage for access control words and specifies this area by an identification word. An I/O operation wherein this identification word is given to the computer by the peripheral device is known as externally specified index (ESI).

ESI completion activity – An ESI completion activity is created when a real-time program initializes a line terminal. The ESI completion activity is activated upon the detection of an ESI interrupt for the line associated with its line terminal group and operates as the highest level activity in the system while processing that interrupt.

Executive or EXEC 8 – Series 1100 Executive System. An Executive is a routine that controls the execution of other routines. The Executive is the principal interface between the user and the system as a whole. It is responsible for such functions as time and space allocation of system resources; first-level I/O control and interrupt answering; logging of system accounting data; first-level debugging assistance; and protection against undesired interaction of users with other users of the system.

Executive control language – The language in which control statements are written.

extended storage – On 1110, 1100/40 Systems, the lower level of main storage, slower than primary storage and occupying the absolute address of 4 million octal and beyond.

external filename – The full name by which a file is identified to the system. In addition to the basic name, full identification may require qualifier and cycle information.

facilities – The peripheral units associated with an application; e.g., tape units, mass storage, printers, etc.

Fieldata – A 6-bit character code which is the native character set of the operating system. The character set and associated codes are listed in Appendix D.

file – An organized collection of data, treated as a unit, and stored in such a manner as to facilitate the retrieval of each individual datum.

granule – The incremental unit of size in which a storage medium can be allocated.

GRS – General Register Stack. The GRS includes 128 program-addressable control registers which consist of 36-bit integrated-circuit registers. The GRS includes indexing registers, accumulators, repeat counters, a mask register, a real-time clock register, interrupt status words, captured timer value, byte manipulation, control registers, and processor state control registers for 1110, 1100/40, 1100/80 Systems.

ICR – Integrated Circuit Register. Same as GRS except used for 1106, 1108, 1100/10/20 Systems.

interlock – A condition by which a peripheral unit is unable to perform an executable command until the condition is removed by the operator.

internal filename – An abbreviated filename used on individual I/O and related operations concerning a particular file. The internal filename may have an implicit association with an external filename, or may be associated to a particular external filename by explicit programmer directive.

IOAU – Input/Output Access Unit. The IOAU controls all transfers of data between the peripheral devices and primary and extended storage on 1110 and 1100/40 systems. Transfers are initiated by a CAU under program control. The IOAU includes independent data transfer paths to primary storage and to extended storage.

IOU – Input/Output Unit. The basic 1100/80 System configuration includes one IOU. The IOU controls all transfers of data between the peripheral devices and main storage. Transfers are initiated by the CPU under program control. The IOU includes independent control paths to the CPU and data transfer paths to main storage. Input/Output (I/O) transmission is through either byte channels or word channels.

ISI – Internally Specified Index. A mode of I/O operation for a SPERRY UNIVAC Series 1100 computer I/O channel, wherein the I/O access to main storage is determined solely by the CPU for 1106, 1108, 1100/10/20 Systems, by the CAU for 1110, 1100/40 Systems, and by the IOU for 1100/80 Systems. This is the normal mode for noncommunications I/O, as opposed to ESI.

language processor – A processor whose principal functions include compiling, assembling, translating, or related operations for a specific

programming language (e.g., COBOL, FORTRAN, Assembler, etc.), as opposed to a system processor.

layered storage – A hardware architecture wherein different parts of main storage have different performance characteristics. The 1110 and 1100/40 Systems have layered storage consisting of primary and extended storage.

LBJ/LIJ/LDJ – Load bank and Jump/Load I-bank and Jump/Load D-bank and Jump. Each program may be composed of, or associated with, a large number of program or data segments; of these, up to four may be active at any given time. Bank Descriptor selection instructions allow a program to select which segments are among the four that are currently active. The combined hardware/software mechanism for switching banks is commonly referred to as 'LBJ/LIJ/LDJ' after the instructions used to accomplish such switching.

main storage – Main storage provides storage for instruction and data words which are accessible to the processors. In systems with an SIU, the main storage is accessed through the SIU.

mass storage – Auxiliary storage which has random access capability; includes all types of auxiliary storage with the exception of magnetic tape.

Master File Directory – A directory maintained by the Executive to control the retrieval and retention of cataloged files.

MSU – Main Storage Unit. The physical cabinet that contains the directly addressable executable storage of the system.

multiprocessing – The simultaneous execution of multiple activities of one or more programs, by employing two or more CPUs which access a common main storage.

multiprocessor – A system having two or more CPUs. Commonly abbreviated as MP.

multiprogramming – The concurrent (interleaved) execution of two or more programs or activities which reside in main storage. This is accomplished by sharing CPU usage through switching.

noise constant – The size of a record, in characters, to be skipped as a noise record. Applies only to magnetic tape files.

omnibus element – An element of arbitrary format used to store general information in a program file.

Operating System – The Series 1100 Operating System. The entire set of system software available for the Series 1100 which is either a part of, or operates under, the Executive System. This includes the Executive System proper, compilers, utility programs, subroutine libraries, and so forth.

P-register – CPU operating register that contains the address of the instruction currently being executed.

packet – A contiguous set of words that contains information to describe the execution of an operation or function to be performed, typically an ER.

partitioning – The partitioning function indicates the operational status of each central complex unit. These status conditions are available to system software for configuration control. See STU, SPU, and ACU.

PCT – Program Control Table. A special table maintained by the Executive containing the bulk of the control information for a particular run and the program (if any) currently in execution for that run.

PMD – Postmortem Dump. A printout of a program's main storage contents following execution. Also the system processor which produces the printout.

primary storage – On 1110, 1100/40 Systems, the higher level of main storage, faster than extended storage, and occupying an absolute address less than 4 million octal.

private file – A cataloged file that can be assigned and accessed only by runs of a particular project.

privileged instruction – One of a set of machine instructions reserved for use by the Executive. If the execution of a privileged instruction is attempted by a user program, a guard mode fault interrupt occurs.

processor (hardware) – See CPU or CAU.

processor (program) – A program incorporated as an integral component of the operating system. Such programs typically reside in the system library (LIB$) as absolute elements and are invoked in a special standardized manner, but are otherwise treated as ordinary user programs. Processors fall in two broad classes: language processors and system processors.

processor control statement – A control statement used to direct the execution of a processor. Such statements have a standardized format which facilitates specification of parameters typically required by processors, such as element names.

program – Generally a series of instructions, in a form acceptable to a computer, prepared in order to achieve a certain result. In the context of run processing, a program is an absolute element to be executed as a task; may be a processor or a user program.

program file – A specially structured file containing a group of elements, residing on sector-formatted mass storage.

programmed breakpoint – On 1110, 1100/40, and 1100/80 Systems, conditioning of a special CPU register such that an interrupt occurs upon reading, writing, or execution of a specific location or group of locations.

project – An identifier used to classify a run for accounting purposes. May also be used to provide implied filename qualification to avoid confusion of similarly named files of different projects.

PSR – Processor State Register. A privileged register that can be referenced only by the Executive and which controls an absolute main storage location of program I- and D-banks and specified mode of operation of the CPU. The PSR contains two basing fields which in conjunction with a relative address determine an absolute main storage location. On 1110, 1100/40 Systems two PSRs (Main and Utility) control 4 program banks. The 1100/80 has 4 BDRs and a designator register that provide an equivalent to the two PSRs on 1110, 1100/40 Systems. BDRs 0 and 2 are equivalent to the main PSR, and BDRs 1 and 3 to the Utility PSR on 1110, 1100/40 Systems.

PSRM – Main PSR (1110, 1100/40), BDRs 0 and 2 (1100/80)

PSRU – Utility PSR (1110, 1100/40), BDRs 1 and 3 (1100/80)

public file – A cataloged file that can be assigned and accessed by a run of any project.

qualifier – An extension to the basic name of a file, employed to resolve a variety of ambiguous situations. Every file has a qualifier which is normally implied according to system conventions rather than being explicitly stated in references to the file.

quota limit – The Executive can be configured to maintain limits defining the maximum system resources that a run, under a specific account and user-id, is allowed to use. The limits are referred to as quota limits.

real-time processing – A mode of operation in which the system's response to external stimuli is sufficiently fast to influence the process or operation being monitored or controlled so as to obtain a desired result. Generally, real-time processing is under the influence of asynchronous inputs from one or more communications device. Real-time processing may also occur in batch or demand mode (typically batch).

reentrant routine – A routine which is written in such a way that the logic of the routine is not modified by execution. The routine only modifies the data which are provided as external parameters when the routine is called. Such routines may be concurrently used by multiple activities or multiple programs. Note that due to sharing of data, a routine may be reentrant at the program level but not reentrant at the activity level.

relocatable element – An element containing a program part in relocatable binary format, suitable for combination with other relocatable elements to produce an executable program (absolute element). Such elements occur most commonly as the output of a language processor to be input to a collection.

remote site	–	Data terminal equipment that is time, space, or electrically distant from a central site, and capable of information exchange with the central site through some common carrier or transmission scheme; typically a communications device.
run	–	Job. A specified group of tasks prescribed as a unit of work for the system. The run is the largest work grouping treated and manipulated as a unit by the Executive. The tasks of a particular run are executed serially in the order specified by the runstream.
runstream	–	A sequence of data images which, taken as a whole, constitute the specification of a run. A runstream consists of an @RUN control statement, followed by other control statements and data, which direct the performance of individual tasks.
Scatter/Gather	–	Scatter Read/Gather Write. An I/O technique wherein multiple discontiguous buffers in main storage (described by a string of access control words) are read into (scatter read) or written out (gather write) in a single continuous operation involving a contiguous area or block on the peripheral device being accessed.
SDF	–	System Data Format. The standard data format employed by the operating system. Briefly, SDF is a sequential, fixed-block, variable-record format in which records may span blocks.
sector-formatted mass storage	–	Mass storage which is accessible in units of 28 words (one sector). This is the most common mass storage format used on Series 1100 Systems.
SIU	–	Storage Interface Unit. A high speed buffer (cache) memory is used to accelerate access to main storage on the 1100/80 system. The range of buffer storage configuration is from one to eight logically independent buffer segments. A buffer segment contains 4K words of storage and is set-associative having four words per block and four blocks per set. The basic SIU contains 4K words of storage and can be expanded by adding 4K buffer segments giving a maximum of 16K words in an SIU. Two SIUs are required for all systems having more than two CPUs; this allows the maximum of 32K words of buffer storage in a system.
SLR	–	Storage Limits Register. A privileged register which provides program isolation in a multiprogramming environment. The Executive loads the SLR with the program's I- and D-bank limits, so that if a program attempts to access an address outside the program area, a guard mode fault interrupt is generated.
SLRM	–	Main Storage Limits Register (1110, 1100/40) associated with PSRM. Equivalent to the limits portions of BDRs 0 and 2 on the 1100/80.
SLRU	–	Utility Storage Limits Register (1110, 1100/40) associated with PSRU. Equivalent to the limits portions of BDRs 1 and 3 on the 1100/80.

SPU – System Partitioning Unit (1110, 1100/40). The SPU contains the controls and indicators for partitioning the system, and the operator panels for each of the CPUs and IOUs.

SRC – Storage Reference Counters. Control registers available to the Executive on the 1110 and 1100/40, which maintain an accurate count of all CAU references to main storage. Two SRCs exist: one for primary storage and one for extended storage.

standard action – Action performed by the operating system in a particular circumstance, in the absence of explicit user directive.

STU – System Transition Unit (1100/80). The STU contains the controls and indicators for partitioning the system, and the operator panels for each of the CPUs and IOUs.

SUA – Standard Unit of Accounting. SUA is a unit of charge which is a monetary unit. SUAs are accumulated during the execution of a run and the accumulation reflects the actual cost in performing the processing services called for by the run. SUAs are used in billing and in imposing monetary limits on accounts and user-ids.

SUP – Standard Unit of Processing. A unit devised to provide a consistent measure of processing service as viewed by the user program. Input to the calculation of SUPs is weighted such that SUPs will, as nearly as possible, determine elapsed time to perform a unit of work in a serial environment on a unit processor with no overlap of I/O and CPU operations.

swapping – The process of storing low-priority or suspended programs on mass storage to allow main storage space to load other higher priority programs.

switching – The process by which the Executive controls usage. This principally involves determination of which activities of which programs are to be executed on which CPU(s) for how long, and the control functions needed to fulfill that determination. This is also called dispatching.

symbolic element – An element containing information generally in human-intelligible format (typically card images). The most common usage of symbolic elements is as source language to be input to a language processor.

symbionts – A complex of Executive routines providing the user interface with unit record peripherals; e.g., onsite and remote card readers and printers.

system – The total SPERRY UNIVAC Series 1100 hardware/software complex comprising an integrated information processing installation.

system processor – A processor whose principal functions are of a specialized systemic service or utility nature (e.g., the Collector, postmortem dump, etc.), as opposed to language processor.

task	–	A discrete processing step in a run, involving the execution of an absolute element (i.e., a processor or user program). Synonymous with program in run processing contexts.
temporary file	–	A transient file created by, accessible to, and existing within the life of, a single run only.
Temporary Program File (TPF$)	–	A mass storage file assigned automatically by the Executive to each run. As a convenience to the user, in a wide variety of program file and element manipulation operations TPF$ is assumed as the program file in the absence of an explicit filename reference.
track	–	In the context of sector-formatted mass storage, a granule consisting of 64 sectors, each sector consisting of 28 words, giving a total of 1792 words per track. For word-addressable mass storage, a granule consisting of 1792 words.
transparent control statements	–	A subset of the control statement set which is allowed only during Demand Processing and which is used to direct immediate processing. They are identified by an "at" symbol (@@) in columns 1 and 2.
unit processor	–	An application having a single CPU. Commonly abbreviated as UP.
unitized channel storage	–	Main storage which is treated as, and accessed as, peripheral I/O hardware.
user	–	An individual or organization that consumes services provided by the system.
user-id	–	Identifies a particular user (person) of the system. A 12-character string that represents a person's identity.
user program	–	Any program other than a processor. Usually developed by a user; however, certain system software packages operate as free-standing user programs (e.g., FMPS).
word	–	A sequence of bits or characters treated as a unit and capable of being stored in a single main storage location. (A word is represented by 36 bits for the Series 1100.)
word-addressable mass storage	–	Mass storage which is capable of being accessed in units of single words. This is most efficient on hardware having this capability (i.e., flying head magnetic drum or unitized channel storage), but for other cases it is simulated by the Executive.

2.2.2. Abbreviations

ACU	–	availability control unit
ADH	–	arbitrary device handler
ADI	–	arbitrary device interface, the interface between the caller and the ADH
AFC	–	abnormal frame count
ANSI	–	American National Standards Institute
ASCII	–	American Standard Code for Information Interchange
AXR$	–	mnemonic designations and absolute addresses of control registers
BDI	–	bank descriptor index
BDR	–	bank descriptor register
BDW	–	bank descriptor word

Bank control values used in conjunction with the LBJ/LIJ/LDJ mechanism.

BPI	–	bits per inch
CLT	–	communications line terminal
C/SP	–	communications/symbiont processor
CTMC	–	communications terminal module controller
DAS	–	directory allocation sector
D-bank	–	data bank
EF	–	external function
EI	–	external interrupt
EOF	–	end of file
EOI	–	end of input
EOM	–	end of message
EOT	–	end of transmission
ETX	–	end of text
FPI	–	frames per inch
FURPUR	–	file utility routine/program utility routine
GCS	–	general communications subsystem

GRS	–	general register stack
I-bank	–	instruction bank
INFOR	–	internal control statement format
I/O	–	input/output
IOAU	–	input/output access unit
LIB$	–	system absolute library
LC	–	location counter
LT	–	line terminal
LTG	–	line terminal group
LTT	–	line terminal table
MFD	–	master file directory
MP	–	multiprocessor
MSA	–	multi-subsystem adapter
MSU	–	main storage unit
P	–	the program address contained in the P-register
PDP	–	procedure definition processor
PFP	–	program file package
RLIB$	–	system relocatable library
SDF	–	system data format
SGS	–	stream generation statement
SIU	–	storage interface unit
SPU	–	system partitioning unit
SSG	–	symbolic stream generator
STU	–	system transition unit
TPF$	–	temporary program file
TRK	–	track
UP	–	unit processor

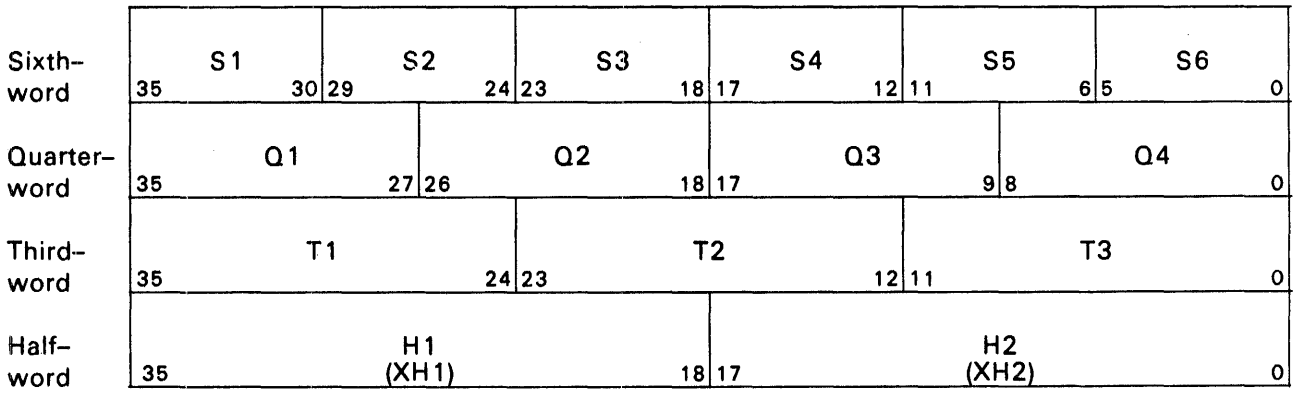
2.3. CONVENTIONS

2.3.1. Notational Conventions

The 36 bits in the Series 1100 computer word are numbered from right to left. For example:

(The mnemonic W is used in ambiguous situations to indicate whole word references.)

To reference partial words, the following mnemonics are used:



The Series 1100 Assembler mnemonics are used whenever machine instructions are discussed. See UNIVAC Series 1100 Assembler Programmer Reference, UP-4040 (current version). The Assembler mnemonics for partial word transfers and registers are defined in SPERRY UNIVAC Series 1100 Assembly Instruction Mnemonics (AIM) Supplementary Reference, UP-9047 (current version).

The mnemonics U and XU are used to indicate immediate data references (operand taken directly from address portion of instruction rather than from the main storage referenced by that address). The mnemonics XH1, XH2, and XU indicate that the 18-bit value is to be loaded with sign-extension to 36 bits (i.e., if bit 17 is a one, bits 35-18 are set to one).

Control registers are referenced by the following mnemonics:

- A0, A1,...,A15 — Accumulators (A0 - A3 also usable as index registers).
- A15+1, A15+2 — Additional accumulators involved in double- or triple-register instructions.
- X1, X2,..., X11 — Index registers.
- R1, R2,..., R15 — R-registers.

Activities may use one of two sets of control registers:

- Major Set — All control registers as given above.
- Minor Set — Registers X8 through X11, A0 through A5, R1, R2, R3.

Subroutines frequently use a subset of the minor set, called volatile registers (i.e., data left in these registers may not be saved). This subset includes:

Registers X11, A0 through A5, R1, R2, R3

The dollar sign ($) is generally used in system-defined external symbols, procedure names, and filenames; to avoid duplication, the user should not use this character. The $ generally occurs as the last character of a symbol, excepting procedure names in which it is usually the second character.

In packet and table formats, parameters in regular type indicate information that must be supplied by the programmer; parameters in italics indicate information that the system returns. Brackets ([ ]) are used to indicate optional parameters.

The symbol Δ is used to indicate a blank character.

In control statement, Executive Request, and procedure call formats, capital letters represent themselves and must be coded as shown. Lower case letters represent variables which must be coded as directed in the text.

Numbers are represented as in assembler syntax; that is, a leading zero specifies octal.

2.3.2. Control Statement Notation

Control statements have the general format:

```
@label:command,options          parameters . comments
          or
@@command,options               parameters . comments
```

Parameters are given in one or more fields separated by commas. A field may specify a single parameter, or may contain several related parameters given in subfields, which are delimited by slashes (///). An ellipsis (...) indicates that any number of additional parameters, of the same format as the last shown, may be given (e.g., reel numbers of a tape file, elements to be listed, etc.).

See 3.2 for a complete discussion of control statement syntax.

2.4. BASIC CONCEPTS OF RUN CONTROL

2.4.1. Run Initiation

The Executive symbiont complex provides the primary input interface between the user and the system. The symbionts control run input from onsite card readers and remote sites, as follows:

1. In batch mode, the entire runstream is normally buffered to mass storage by the symbionts before run processing is initiated. At this point, an Executive component called the coarse scheduler takes over. It examines that portion of the runstream prior to the first task for initial facilities requirements. Based on those requirements, and certain other operating parameters such as run priority and deadline time (if any), the coarse scheduler determines the proper time to open the run.

2. In demand mode, the run is normally initiated immediately upon acceptance of the @RUN control statement. Additional runstream input generally occurs dynamically on an interactive or conversational basis. See Section 8 for a complete discussion of demand processing.

When a run is opened, two temporary files are automatically assigned to it: the temporary program file (TPF$), and the run diagnostic file (DIAG$) which is not normally referenced directly by the user.

2.4.2. Run Execution

Once a run is opened, the coarse scheduler, in cooperation with the symbiont interface routines, processes the runstream sequentially. When a control statement is encountered, the appropriate Executive routines are invoked to accomplish the specified action. When a control statement that causes execution of a task is encountered, the coarse scheduler sets up the task and passes control to the dynamic allocator (see 2.5) for execution. Runstream images are then passed by the symbiont interface routines directly to the task as data, one at a time as requested by the task, until the next control statement is encountered or the task terminates.

Runstream data images (i.e., images that are not control statements) are bypassed with a warning diagnostic if encountered when a task is not being executed.

2.4.3. Symbiont File Concepts

From the standpoint of run processing, there are two basic classes of symbiont files: primary symbiont files and alternate symbiont files. They differ in usage rather than structure.

Primary symbiont files comprise the standard files through which the user communicates with unit-record equipment. There are three types of primary symbiont files:

1. Primary Input File (READ$). This file is automatically established for each run and contains the runstream (see 2.4.1 and 2.4.2). This file cannot be manipulated, as a unit, by user directive. In demand mode, this file is the terminal device.

2. Primary Print File (PRINT$). Each run has associated with it a standard output print file. In general, all control statements, Executive diagnostic messages, and summary accounting information are printed in this file, as well as primary print output generated by the tasks of the run.

In the normal case, the Executive establishes and controls the disposition of the primary print file without the need for user directives, as follows:

a. In batch mode, the file is buffered on mass storage. At run termination, it is printed at the site from which the run was initiated.

b. In demand mode, print output occurs at the terminal as it is generated with buffering limited to about 100 lines.

The primary print file may be thought of as an output stream. Using a procedure called breakpointing, the user may direct this stream to the files and/or partition it into several Executive files called parts. The use of breakpointing allows printing of large volume output to begin prior to run termination. To simplify file referencing, the current primary print file is always referenced by the generic name PRINT$, regardless of whether the file is an Executive or user file.

3. Primary Punch File (PUNCH$). Primary punch output is handled in the same fashion as primary print output, with two exceptions:

   a. No file is established unless user tasks generate primary punch output.

   b. Punch output generated at demand terminals will be directed to the device determined by the output device association group for those terminals in the absence of user directives.

   The current punch output file is always referenced by the generic name PUNCH$.

The user may define alternate symbiont files in addition to the primary symbiont files. The principal purpose of this feature is to allow the user multiple concurrent symbiont operations of a particular type (read, print, or punch).

Symbiont concepts and interfaces are discussed in detail in 3.6 and Section 5.

2.4.4. Run Termination

A run terminates upon reaching the end of the runstream (@FIN control statement) or as the result of an abnormal task termination. A number of actions are triggered at run termination, normally including:

- Summary accounting information is entered in the print file, including such items as run start and termination time, SUPs used, pages printed, pertinent console messages, and so forth.
- Symbiont output files are closed, and queued for printing/punching in batch mode.
- Any main storage space allocated to the run is released.
- All facilities allocated to the run are released back to the system facilities pool, with the exception of mass storage space being retained in a cataloged file.
- Temporary files are freed and cease to exist.
- Cataloged files are freed and, in the absence of assignment options to the contrary, retained. Any exclusive use interlocks are released.

2.5. BASIC CONCEPTS OF TASK CONTROL

The primary responsibility for the execution of individual tasks (programs) belongs to two Executive components: the dynamic allocator, which manages main storage, and the dispatcher, which allocates CPU usage. Of these, the dynamic allocator is dominant in the sense that a task cannot use CPU time unless it is loaded in main storage.

Normally, the Executive executes a number of tasks concurrently (unit processor) and/or simultaneously (multiprocessor) by time-sharing the usage of main storage and CPU time. However, in most cases the user is unaware of, and need not be concerned with, the presence of other tasks in the system.

2.5.1. Real-Time Mode

Runs and tasks are always initiated in demand or batch mode. Real-time mode is entered only when a task requests it.

Most operational details of Executive task control are of interest primarily in real-time applications. These are covered in Section 10. The following paragraphs apply only to demand and batch tasks and activities.

2.5.2. Task Initiation

The dynamic allocator initiates a task by allocating sufficient main storage space to accommodate the program, loading the program into main storage, assigning it a single activity with the major set of control registers and setting that activity's current instruction address (P-register) to the program starting address specified at collection.

2.5.3. Task Execution and Switching

Once the task is ready for execution, its initial activity is passed to the dispatcher. Eventually, that activity is given control and allowed to execute instructions for a period of time until it voluntarily relinquishes control (for example, to do I/O), or has used up the time allotted it by the dispatcher, or is preempted by a higher priority activity, at which time the dispatcher switches to another activity (if there is one requiring CPU service). When the original activity's turn comes again, its CPU environment is restored and it resumes execution at point of interrupt. This switching process continues until the activity terminates. The same process also applies to any additional activities registered by the program.

A task remains loaded in main storage until it terminates, or until it must be removed (swapped) in favor of a higher priority task or because its space requirements have increased. In the case of swapping, the dynamic allocator first suspends the program by deactivating all of its activities such that the dispatcher does not attempt to return control to the activities. The program's main storage contents are then written to mass storage. When the dynamic allocator determines that the program should be reloaded, it is read back into main storage and its activities are reactivated (made candidates for switching).

Although batch programs may be swapped, swapping occurs most commonly for demand programs, which need not be in main storage while awaiting input from the demand terminal user.

Note that a program is not necessarily always executed in the same place in main storage, and is generally unaware of where it is loaded. This is accomplished by means of hardware relocation using basing registers. Whenever a program is loaded (or reloaded) into main storage, the dynamic allocator

sets the basing register values associated with each activity to reflect the program's absolute position in main storage. The program itself uses program-relative addresses which the hardware adds to the appropriate basing register to determine the true absolute main storage addresses of the program's operands and instructions.

2.5.4. Executive Requests

A program activity accomplishes its functions in two basic ways. The most common, of course, is by executing instructions; the other is by having the Executive do the actual execution. This is done via the Executive Request (ER) instruction.

The ER is the principal interface between an executing program and the Executive. It has the same fundamental relationship to task control that control statements have to run control. ERs are provided for a wide variety of functions, including activity control, input/output, facilities control, clocking, storage control, etc.

ERs related to specific areas of the Executive (e.g., I/O, symbionts, etc.) are covered in associated sections. Section 4 contains a more detailed discussion of the ER mechanism, certain ERs not covered elsewhere, and a master cross-reference for all ERs.

2.5.5. Multiprogramming Considerations

The basic execution unit is the activity. An activity is defined as a logical CPU, with either a major or minor register set, an instruction counter, and specified storage limits. The initial, or any other activity of a program, may create any number of new activities. No hierarchy within activities is imposed or recorded with the system.

Programs which do not register multiple activities in general need not be concerned with the impact of switching. In essence, such programs can be written as if they have a single CPU to themselves.

Multiactivity programs, in general, require semaphore, queuing, and special contingency facilities for correct and efficient operation. These facilities are made available by the test and set (semaphore) hardware mechanism, test and set queuing, and contingency registration including special common bank contingency mechanisms.

For demand and batch programs, the dispatcher treats all activities equally, insofar as no activity can be certain of executing ahead of any other activity unless the programmer employs some method of synchronization; interrupt activities are given priority, but only for a short period that is not programmer controllable.

Two basic approaches to activity synchronization are provided:

1. A set of ERs which allows activities to wait for events triggered by other activities (see Section 4).

2. Use of the Test and Set (TS) hardware instruction, which is designed expressly for synchronization of asynchronous processes. This instruction functions, in conjunction with the dispatcher, as follows:

    a. If bit 30 of the operand is zero, the next instruction is executed.

    b. If bit 30 is a 1, an interrupt occurs and the activity's switching priority is reduced. The activity receives control back at the TS instruction on its next turn to execute; if bit 30 is still set, the process is repeated. In this fashion, the activity spirals downward in priority until bit 30 is reset to 0 and execution then proceeds to the next instruction.

c. Regardless of the setting of bit 30, bit 30 is set to 1, bits 35–31 are cleared to zero, and bits 29–0 are unaltered.

d. The testing and setting of bit 30 occurs in a single main storage reference, thus ensuring that two CPUs cannot both find bit 30 set to zero.

When the protected sequence has been completed, a simple Store Zero (SZ) instruction clears the test and set condition. The instruction sequence for protecting against concurrent execution of a critical instruction sequence is:

```
TS      IND
CRITICAL SEQUENCE
SZ,S1   IND
```

where:

IND is any main storage location; it may be associated with a particular data set to be protected or may be a global lock for the sequence. It must have had S1 initialized to zero.

The 1110, 1100/40, 1100/80 have two additional synchronization instructions which operate in a fashion similar to TS, but which do not interrupt. They are the Test and Set and Skip (TSS), and Test and Clear and Skip (TCS).

2.5.6. Task Termination

A task terminates when all of its activities have terminated. On termination, main storage space for the program is released. On a normal termination, the coarse scheduler continues processing the runstream.

An error termination occurs when one or more activities terminate in error. In this case, further batch runstream processing is usually limited to, at most, the processing of a postmortem dump (PMD). Demand mode users are given an appropriate error message.

All ERs are subject to extensive validation to ensure against interference with other runs. In most cases, when an error is found, the activity is placed in error mode. This does not necessarily mean that the activity actually terminates, but rather that a contingency occurs; standard action is error termination in the absence of a user specified contingency routine.

See 4.3.2 and 4.9 for details on activity termination and contingencies.

2.5.7. Program Protection

The multiprogramming and multiprocessing capabilities of the Executive System imply that many programs (including the EXEC itself) are simultaneously resident in main storage and in various stages of execution at any point in time. A basic design objective of the system is to provide a well structured protection mechanism to prevent unintended interactions between these programs. Programs may interact through shared (protected or writeable) main storage, or by other means. However, it is intended that any interaction be controlled. Control is exercised by the use of hardware features which define privileged (EXEC) and user states in each CPU and which restrict main storage access by defined main storage limits.

When a CPU is operating in user mode, the executing program cannot reference main storage outside its defined address space nor can it execute input/output or other privileged instructions. All I/O and

other services must be carried out through service calls to the EXEC. The EXEC validates all calls to ensure that they do not violate the addressing and other privileges currently available to the user program. A number of levels of privilege are available to user programs, but validation of user calls on the EXEC is always carried out so that faulty user programs should not impair the operation of the system.

If multiple users share the same write-protected main storage there is no interaction between the programs and any action or failure of one program will not affect any of the other programs. If two or more programs are sharing writeable main storage, then each program must access and modify the shared main storage by agreed means to avoid conflict or errors. The test-and-set (TS) and TS Queuing mechanisms are provided to facilitate such interactions.

2.5.8. Address Space Sharing

Since each user program may have at any time two (on 1106, 1108, 1100/10, and 1100/20 Systems) or four (on 1110, 1100/40, and 1100/80 Systems) independently addressable active storage segments, one or more of these may have been defined as a shared or Common Bank. If a Common Bank is referenced by a program, the reference is to the one common physical copy of that bank, regardless of how many programs are currently referencing the bank. Such Common Banks need not be write protected.

There are two frequent uses of Common Banks. One is the use of write protected Common Banks to contain frequently used pure code. The other use is to provide an efficient means of inter-program communication and data transfer.

2.6. FILENAMES AND ELEMENT NAMES

2.6.1. Filenames

Each file in the operating system is assigned a unique name to distinguish it from all other files. The filename is required in the many control statements and directives that are used to reference files.

The following notation is used to specify filenames:

[[qualifier]*]filename[(F-cycle)] [/[read-key] [/write-key]]

Filename is used as the basic name of the file, and qualifier is used to establish uniqueness between files that have the same basic name. F-cycle is used to identify a particular file from a set of related files that have the same qualifier and filename (see 2.6.3). Read-key and write-key are used to obtain read and write access, respectively, to the file. These keys are not a part of the filename for purposes of assigning a file (see 3.7).

Qualifier and filename each consist of from 1 to 12 characters selected from the set: A-Z, 0-9, -, and $ (see 2.3.1 for caution on the use of $). F-cycle is an integer number (see 2.6.3 for range of values). Read-key and write-key each consist of one to six characters; any Fieldata character may be used except blank, comma, slash, period, and semicolon.

All parameters, except filename, are optional when naming a file. For those parameters that are omitted, the Executive provides standard default. Whenever the qualifier is omitted but the * is specified, the qualifier specified on the last @QUAL control statement (see 3.7.6) is used; however, when no @QUAL control statement is given or if both the qualifier and the * are omitted, the project-id from the @RUN control statement (see 3.4.1) is used as the qualifier. If the read-key or write-key is omitted, blanks are supplied as the key. If F-cycle is omitted when naming a set of files, the relative cycle number of -0 is supplied (see 2.6.3).

2.6.2. External and Internal Filenames

The term 'external filename' refers to the name of a cataloged file or to a temporary file assigned to a particular run. As previously stated (see 2.6.1), it may be necessary to specify the qualifier, filename, and F-cycle to ensure that the file is uniquely identified. In the case where the file is uniquely identified, but filename is not unique (either qualification or cycling has caused unique identification), it is necessary to attach a unique filename (internal filename) to accomplish I/O or related operations. It may also be convenient to have a short name by which to refer to a file that has a long external filename; or a standard name which is attached to the particular file assigned or to be assigned to the run. By means of the @USE control statement (see 3.7.5), the Executive provides the capability of attaching such an alternate or 'internal filename' to a file for referencing within a run. For example, the filename EZ can be made the internal filename for file ABCDEFGHIJLM* MLJIHGFEDCBA(-23) by the control statement:

```
@USE  EZ,ABCDEFGHIJLM*MLJIHGFEDCBA(-23)
```

For the remainder of the run, whenever a reference to the file is to be made, EZ can be used as the filename. Thus the file can be referenced by either the external or internal filename. Several internal filenames can be attached to a particular external filename by multiple @USE control statements. For example:

```
@USE  EZ,ABCDEFGHIJLM*MLJIHGFEDCBA(-23)
@USE  BACKUP,EZ
@USE  Q,BACKUP
```

The file can now be referenced by any of the following filenames:

```
ABCDEFGHIJLM*MLJIHGFEDCBA(-23)

EZ

BACKUP

Q
```

The internal filename established is valid only for the particular F-cycle of the external filename. Thus, the internal filenames established in the foregoing example are valid only for cycle -23 of the set of files.

The following is an example of the use of internal filenames for referencing files with nonunique filename parameters:

```
@USE  NEWCYC, ID*FILESET(0)
@USE  OLDCYC, ID*FILESET(-1)
@USE  OTHER, QUAL*FILESET
@USE  SAME, UNIQUE*PF
```

The internal filenames NEWCYC and OLDCYC may be used to accomplish I/O on different cycles in a set of files. The name OTHER is used to distinguish the file QUAL*FILESET from all other files with filename FILESET. Assuming that no other file with filename PF is used in the run, filename PF or SAME may be used for I/O and related references to uniquely identify the file to be accessed.

Internal filenames may contain 1 to 12 characters from the set: A-Z, 0-9, -, and $.

2.6.3. File Cycles (F-Cycles)

The use of F-cycles enables the user to manipulate any of a set of cataloged files without modifying the runstream.

Each qualifier*filename constitutes an F-cycle set. Each file within one of these sets of cataloged files has the same read and write keys as well as qualifier*filename.

As a file of a given F-cycle set is being created, a number is assigned to it. This number is called the absolute F-cycle number and uniquely identifies that particular file along with its qualifier*filename (see 2.6.1). Absolute F-cycle numbers are unsigned integers that begin with 1 and continue through 999, at which point the numbering recycles to 1. The circular assignment of F-cycle numbers does not cause conflicts since a maximum of 32 consecutively numbered files may be retained in a set.

A file within an F-cycle set may be referenced by its absolute F-cycle number or by a relative F-cycle number. Relative F-cycle numbers are signed integers in the range +1 to -31. Relative F-cycle number 0 (-0 or +0) refers to the cataloged file whose absolute F-cycle number is highest. Relative F-cycle -1 refers to the next highest, etc. The highest is defined as having the highest absolute F-cycle number of any of the files in the F-cycle set; unless, of course, the absolute F-cycle numbers are at the point of recycling from 999 to 1, in which case 1 is higher than 999.

To create the next sequential absolute F-cycle the relative specification +1 can be used. When this +1 file is cataloged (by freeing the file or by run termination) its relative F-cycle number is set to 0 and other existing files of the set have their relative F-cycle numbers decreased by 1. When a file is deleted from a set, the relative F-cycle number of those files with a more negative relative F-cycle number will have their relative F-cycle number increased by 1, thus maintaining consecutive relative numbering.

When the maximum number of files in one F-cycle set is exceeded, the record of the file with the most negative relative F-cycle number is deleted and, if the file is stored on mass storage, the file's text is deleted.

The deletion of the last remaining file of a set causes all recognition of the set to be removed from the system.

As an example of the correlation between the absolute and relative F-cycle numbering schemes, assume that the most recently cataloged cycle of a file has an absolute F-cycle number of 28. When referring to this file, the F-cycle could be given as 28 or -0. Now assume a new cycle of the file is to be cataloged. Since the file has not yet been cataloged, its relative F-cycle is +1 and its absolute F-cycle is 29. When the new file is freed (cataloged), the absolute F-cycle remains as 29 but the relative F-cycle becomes 0. The file whose absolute F-cycle is 28 is now given a relative F-cycle of -1; the file whose absolute F-cycle is 27 is now given a relative F-cycle of -2; and so on down the line. The use of relative F-cycle numbers enables the user to refer to a particular relative backup file.

For example, assume that the nature of the program requires that a particular file always be created from a file four cycles earlier than the file being created and later cataloged. The relative F-cycle of -3 could be used when referencing the earlier file, and this reference would be valid each time the program is run. If an absolute F-cycle notation has been used, the F-cycle designation must be changed each time the program is run.

2.6.4. Element Names

Four different types of elements are recognized by the operating system:

1. symbolic elements, (including PROC elements),
2. relocatable elements,
3. absolute elements, and
4. omnibus elements.

Typically, symbolic elements contain source language images for language processors or Executive control statements and data to be processed by other control statements. Relocatable elements are the binary output of certain processors such as COBOL and FORTRAN. Absolute elements are the output of the Collector. Omnibus elements are arbitrary format elements used to store any kind of information as a program file element.

Elements of any or all of the four types are contained in program files (see Volume 3–11.2.1) on mass storage or in element files on tape. Each element is uniquely identified within a file by its element type and element name. Of course, each file in the operating system has a unique name and the filename is used as part of the element name to uniquely identify each element. The following notation is used to specify element names:

[[filename].] element-name[/version] [(cycle)]

Filename is the name of the file in which the element is contained; filename conforms to all the rules specified in 2.6.1. Element-name is used as the basic name of the element. Version is used to establish uniqueness between elements in the same file that have the same type and element-name. Cycle is used to specify a particular update of a symbolic element; cycles are not used for relocatable, absolute, and omnibus elements.

Element-name and version may consist of 1 to 12 characters selected from the set: A–Z, 0–9, –, and $ (see 2.3.1 for caution on use of $). Cycle is an integer number (see 2.6.3 for range of values). All parameters, except element-name, are optional when naming an element. For those parameters omitted, the Executive supplies values according to standard rules. If cycle is omitted for a symbolic element, relative cycle –0 is supplied in order to select the most recent cycle. If version is omitted, blanks are supplied. Omission of filename and the period implies reference to the run's temporary program file (TPF$). Filename must refer to a program file, unless otherwise stated. In a series of element names such as on a processor call statement (see 3.9), the period may be given without filename to specify the same file as for the preceding element in the series.

2.6.5. Symbolic Element Cycle

To save altered or deleted images (updates) within symbolic elements, an integer parameter called "cycle" is associated with each symbolic element. New cycles are created by specifying the U option on the processor call statement (see 3.9).

Each item (image) in a symbolic element has a cycle number that indicates to which element cycle it was entered, and, if deleted, a delete cycle number to indicate in which cycle the item was deleted. When an element is updated, the added items are inserted in their proper position and they are given an entered cycle number one greater than the last cycle of the element. Any items deleted by the update are given a deleted cycle number one greater than the last cycle of the element.

When specifying a symbolic element for compilation or assembly, the user may select a specific update from a sequence of retained updates by referencing the proper cycle number as part of the element name.

A system-standard maximum of five consecutively numbered cycles may be retained in a symbolic element. This maximum may be set to any value needed (up to 63) for a particular element by the use of the @CYCLE control statement (see Volume 3 – 4.2.1.16). As soon as the number of retained updates for an element exceeds the specified maximum, the update with the lowest numbered cycle is combined with the update having the next higher cycle number to create a new cycle which in effect becomes the oldest cycle of the element.

A particular cycle may be referenced by either an absolute or a relative cycle number. Absolute cycle numbers are unsigned integers in the range 0 to 62; however, since only a limited number of cycles are retained, the absolute cycle numbers used when referencing an element must be in the range of those absolute cycles retained. Relative cycle numbers are signed integers. If the relative cycle is given as –n, then absolute cycle r–n is referenced, where r is the most recent absolute cycle retained. If +n is used, then absolute cycle x+n is referenced, where x is the oldest absolute cycle retained. The use of relative cycle numbers makes it unnecessary to know the absolute cycle number of either the oldest or most recent cycle retained.

Since absolute cycle numbers may not be greater than 62, when absolute cycle 62 of an element is updated, all retained cycles are renumbered. The renumbering assigns cycle 0 to the oldest cycle retained, 1 to the next oldest, and so forth. For example, assume that a maximum of three cycles may be retained, cycles 60, 61, and 62 are currently retained, and cycle 62 is to be updated; as a result of the update, the element would contain cycles 0, 1, and 2. Cycle 0 is the equivalent of the previous cycle 61, 1 is the equivalent of 62, and 2 is the update of cycle 62; cycle 60 was dropped since a maximum of three cycles may be retained.

The technique of using cycled symbolic elements offers two distinct advantages over other methods:

1. The equivalent of many different copies of the same element can be kept while requiring very little additional storage space over that needed for a single copy.

2. Earlier copies of the element can be referenced without having to prepare correction cards to delete later corrections. If, however, any cycle other than the latest cycle is corrected and the corrected cycle is to be retained, all cycles following the cycle to be updated are deleted. The new cycle number is the updated cycle number plus one.

2.6.6. Referencing Files and Elements

Many of the control statements and directives discussed in this manual require that the particular file or elements desired be specified. If the control statement or directive specifies *filename*, the following form is used:

qualifier*filename(F–cycle)/read–key/write–key

If *eltname* is specified, the following form is used:

filename.element–name/version(cycle)

If *name* is specified, either a filename or element name may be used.

On certain control statements, such as those of the FURPUR processor (see Volume 3 – Section 4), if a filename is expected but the field is left empty, reference to the run's TPF$ is assumed. On other control statements, such as the processor call statement (see 3.9), if an element name is expected but the file is left empty, the name TPF$.NAME$ is supplied.

Note that if all optional parameters are dropped from a filename or eltname, it would seem that the operating system could not distinguish between a file and element name where either may be used. In such cases, the absence of a period in the specification implies that it is an element name. Thus:

name	references an element (in TPF$);
.name	references an element (in some implied file);
name-1.name-2	references an element (name-2) in a specific file (name-1);
name.	references a file.

The use of the period to distinguish a file from an element name is required only when either of them could be specified. In some control statements and directives, only a filename is required and the Executive assumes that any name specified is a filename. Confusion may be avoided by always giving a period following a filename. Distinction between the four types of elements is done either by context or by option letter specification.

2.6.7. Examples of File and Element Reference

Name		Interpretation
PAYROLL*BACKUP(-2).	-	References relative F-cycle -2 of file BACKUP with a qualifier of PAYROLL
COST*PROG.EDIT	-	References element EDIT in file COST*PROG
*BACKUP.TLU/TWO	-	References version TWO of element TLU in file BACKUP. Qualifier is taken from the last @QUAL control statement encountered, or the project-id parameter if no @QUAL control statement was used.
PCF6.INTL(14)	-	References absolute cycle 14 of element INTL in file PCF6. Qualifier is taken from project-id parameter in @RUN control statement.
SORT	-	References element SORT in TPF$ file.
.SORT	-	References element SORT in TPF$ file.
SORT.	-	References file SORT.
SORT(-4).	-	References the fifth most recent F-cycle of the F-cycle set of file SORT.
SORT(7)/YES/NO.	-	References F-cycle 7 of file SORT. The read and write keys are YES and NO, respectively.
MERGE//IN.	-	References file MERGE. Write key is IN and read key is blank.
MERGE/IN.	-	References file MERGE. IN is the read key.
A*B(-1)/CQ/QT.C/D(-3)	-	References the fourth most recent cycle of version D of element C in the second most recent F-cycle of file A*B. Read and write keys are CQ and QT, respectively.

NOTE:

When an F-cycle or element cycle is not specified, it is assumed that the most recent (newest) cycle is desired.

SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Series 1100

Executive System
Volume 2
EXEC Level 36R2

Programmer Reference

UP-4144.23-A

This Sperry Univac Series 1100 Library Memo announces the release and availability of Update Package A for "SPERRY UNIVAC Series 1100 Executive System, Volume 2, EXEC Level 36R2, Programmer Reference," UP-4144.23. This is a Standard Library Item (SLI).

This update package corrects Table D-5 by including the Octal/Decimal Conversion Table information which was inadvertently omitted from UP-4144.23.

The inclusion of Update Package A continues to reflect Series 1100 Executive level 36R2. The Software Release Documentation accompanying each new level of the Executive will contain the information needed to modify the current UP documentation. The Series 1100 Executive level 36R2 software described in this document is a proprietary product of Sperry Univac. As such, authorization to receive the software package must be established by contractual agreement prior to transmittal.

Copies of Update Package A are now available for requisitioning. Either the update package alone, or the complete manual including the update package may be requisitioned by your Sperry Univac representative. To receive the update package alone, order UP-4144.23-A. Your manual should be updated in accordance with the instructions on the Update Summary Sheet provided with the package. To receive the complete manual including the update package, order UP-4144.23.

To Mailing Lists AC, BZ, CZ, and MZ.

Update Package A for UP-4144.23 (Library Memo, 11 pages, cover, and Update Summary Sheet) to Mailing Lists 8, 9, 9U, 31U, 37, 37U, 38, 62, 63, 63U, 64, 64U, 81, 81U, 83, 83U, 89, and 89U.

Library Memo for UP-4144.23-A

January 1980

3. Executive Control Statements

3.1. INTRODUCTION

Control of the operating environment for the Series 1100 is accomplished through a set of Executive control statements. These control statements direct the Executive System in the processing of a run. Control statements may invoke Executive functions such as scheduling, assignment of facilities, and so forth; or may cause the execution of a user program or a processor (i.e., a task). The Executive control statements are compact and descriptive to facilitate use and yet provide full access to all of the features and functions of the Executive System.

3.2. CONTROL STATEMENT FORMAT

Control statements with the exception of transparent control statements, consist of a recognition character, (@), and four major sections:

- the label field
- the operation fields
- the operand fields
- comments

The transparent control statements consist of two recognition characters, (@@), followed by only the operation and operand fields (see 3.2.8).

Each of these sections may contain one or more parameter fields and each of the parameter fields may be further subdivided into parameter subfields. The recognition character is the at symbol @, which is a 7–8 punch for punched cards or its equivalent for other devices. The recognition character must always appear in column 1. The format of the control statements, with the exception of the @END, @ENDCL, @ENDF, @EOF, and @FIN control statements, is free-form; that is, the order of the parameter fields within the control statement is fixed, but the parameter fields are not restricted to a particular column. The aforementioned control statements must be coded exactly as shown in their respective descriptions.

The maximum length of a control statement is 14_{10} words (84 characters) per line. In batch mode, images exceeding this length cause termination of the run. A control statement requiring more than one line or card may be continued (see 3.2.5).

The basic format of a control statement is:

	Label Field	Operation Fields	Operand Fields	Comments
@	label:	command,options	parameter-field-1,...,parameter-field-n	comment

3.2.1. Label Field

The label field need not appear but may be used to name a control statement. The label is limited to six characters from the set: A–Z, 0–9. The first character of a label must be alphabetic. If a label is specified, it must be followed immediately by the colon (:). A label is used only when dynamic adjustment of the runstream is required. The discussion of its use is in 3.10.3.

3.2.2. Operation Fields

Unless the control statement is to be used only as a label statement (see 3.10.3), the command field must always be specified, as it determines the control statement's basic operation. The command on all control statements except the processor control statements (see 3.9) is limited to six characters from the set A–Z and 0–9, the first of which must be an alphabetic. The command field is terminated by a blank or, if options are specified, by a comma.

The options field specifies the options in the form of unsequenced alphabetic characters related to the particular command. On some control statements, the options can be broken into subfields, each of which is separated by a slash (/). A blank character or a series of blank characters separates the operation fields from the operand fields.

3.2.3. Operand Fields

The operand fields specify parameters associated with the command fields. These are separated by commas. The content of each operand field, the number of operand fields, and whether each is required or optional varies with the command selected. Operand fields, in turn, may contain parameter subfields that are separated by a slash (/) or other predefined characters. For the most part, these subfields are optional within a field. Thus, it is possible to specify parts of a field without specifying the entire field.

3.2.4. Control Statement Annotation

Control statements may be annotated with comments following the operand field. At least one blank character must precede the comment. The comment itself may contain any character except the semicolon (;), which is the continuation character. The comment is terminated by the end of the card or its equivalent for other input devices. The comment is never required. If the operand parameter fields are omitted, the comment must begin with a blank followed by a period (.) followed by a blank. This is also true when the content of an operand parameter field is unrestricted and variable in length (as with the @LOG and @MSG control statements). The @XQT control statement is an example of a statement where operand parameter fields are possible but may be omitted.

3.2.5. Control Statement Continuation

In certain situations, a control statement may require more than one line or card. In such cases, coding a semicolon (;) indicates continuation on the next card or line. A control statement may be split at any point where a leading blank is allowed. It is treated logically as a space. Continuation of the next line can begin in any column, with one exception: an at character (@) must not be placed in column one of the continuation line.

NOTE:

A filename or element name is legal for certain control statement fields (see 2.6.6). In this case a filename cannot be continued at any of its subfields. If the period to distinguish the filename is not encountered before the continuation character, an element name is assumed.

3.2.6. Leading Blanks in Fields

Leading blanks within a statement are permissible in the following cases:

- Following the at (@) character (except as noted in 3.2)
- Following a colon (:) when a label is specified
- Following a parameter field separator (,)
- Following a parameter subfield separator (/) and other predefined characters (also when it is a subfield separator)
- Within the comment field

Example:

```
@PRT,T FILENAME.ELEMENTNAME
```

A blank, placed at any position in the coding other than those listed, is interpreted as the termination of the image.

3.2.7. General Default Rules

When parameter fields and subfields are optional, the following rules apply, where an empty field is defined as one that contains no nonblank characters:

1. Parameter field separators must be specified, left to right, through the last parameter given; fields preceding the last parameter may be empty; trailing field separators need not be specified.
2. The same holds true of parameter subfield specifications within a field.

For example, in the magnetic tape @ASG control statement, the only required parameters are ASG and filename (see 2.6.1). The format of this control statement is:

```
@label:ASG,options    filename,type/units/log/noise/processor/tape/format/dc,reel-1/
                      reel-2/.../reel-n,expiration period
```

The following example illustrates the possible coding combinations that result from applying the general default rules:

```
@ASG,A          FILENAME,,REEL1
@ASG             FILENAME,20N///4
@ASG,T           FILENAME,20N//X,REEL1/;
                 REEL2/REEL3
@ASG,A           FILENAME,///6
@TAG1:ASG        FILENAME,,REEL1
```

Although control statements are free-form, most programmers align the label, operation, and operand fields.

3.2.8. Transparent Control Statements

In both batch and demand processing, data images and control statements in a runstream are processed sequentially and only upon request by the Executive or by a program operating in that run. However, a special mode of processing control statements during demand processing is called transparent mode. This mode directs the Executive to process a control statement immediately after it has been input from a remote inquiry terminal. The processing called for by the control statement is also done independently of any current program execution or control statement processing in the run stream. If, however, the processing called for causes the run to be placed in error termination, all run activities are aborted. This mode of executing a control statement is specified by a special character, a second @ in column 2 on the control statement. Control statements which can direct or specify this mode of operation are called transparent control statements.

The processing called for by the control statement is also done independently of any current program execution or control statement processing in the run stream. If, however, the processing called for causes the run to be placed in error termination, all run activities are aborted.

Transparent control statements are a subset of the control statement set. The syntax rules for normal control statements, with the following exceptions, also apply to transparent control statements. The exceptions are:

1. The identification of a transparent control statement consists of an @@ versus an @ for a normal control statement.

2. The use of a label on a transparent control statement, while not prohibited, is meaningless.

The rules governing the use of transparent control statements are specified in 8.1.1.3.2.

3.3. SUMMARY OF CONTROL STATEMENTS

The Executive control statements can be divided into eight groups. These groups are:

1. Scheduling statements

2. Message statements

3. Symbiont directive statements

4. Facility statements

5. Data preparation statements

6. Program execution statements

7. Dynamic runstream modification statements

8. Checkpoint and Restart statements

Table 3-1 lists all the Executive control statements in their respective groups and presents a brief description of each statement's function.

In addition, the following transparent control statements perform the same functions as the associated control statements described in the table: @@PASSWD, @@START, @@LOG, @@MSG, @@HDG, @@SYM, @@BRKPT, @@ASG, @@MODE, @@CAT, @@FREE, @@USE, @@QUAL, @@CKPT.

Certain control statements, because of their complexity or their close association with concepts discussed elsewhere in this manual, are not discussed in this section. Table 3-1 contains references to the sections in which these statements are discussed.

Table 3-1. Summary of Executive Control Statements

Statement Group	Control Statement	Description
Scheduling Statements	@RUN	Appears at the beginning of each run. Provides accounting, scheduling, and run identification information (see 3.4.1).
	@FIN	Appears at the end of each run (see 3.4.2).
	@PASSWD @@PASSWD	Appears anywhere between @RUN and @FIN. Provides the system with the user's password. Also, allows the password to be changed (see 3.4.1).
	@START @@START	Initiates the execution of an independent run (see 3.4.3).
Message Statements	@LOG @@LOG	Places user specified information in the system log (see 3.5.2).
	@MSG @@MSG	Places a message on an operator's (central site) console (see 3.5.1).
Symbiont Directive Statements	@HDG @@HDG	Places a heading line on print output (see 3.6.1).
	@SYM @@SYM	Directs nonstandard symbiont output action (see 3.6.3).
	@BRKPT @@BRKPT	Segments primary symbiont output files and closes alternate symbiont files (see 3.6.2).
	@COL	Specifies various forms of image input (see 3.6.5).

Table 3-1. Summary of Executive Control Statements (continued)

Statement Group	Control Statement	Description
Facility Statements	@ASG @@ASG	Assigns files to a run. There are four types of @ASG control statements: sector-formatted mass storage (see 3.7.1.1). tape (see 3.7.1.2). word-addressable mass storage (see 3.7.1.3.1). absolute device (see 3.7.1.3.2).
	@CAT @@CAT	Catalogs mass storage or tape files (see 3.7.3).
	@FREE @@FREE	Unassigns a file and its input/output device or mass storage area (see 3.7.4).
	@USE @@USE	Sets up a correspondence between internal and external filenames (see 3.7.5).
	@QUAL @@QUAL	Defines a filename qualifier (see 3.7.6).
	@MODE @@MODE	Changes the mode settings (density, parity, etc.) of tape files (see 3.7.2).
Dynamic Run Stream Modification Statements	@ADD	Dynamically expands the runstream (see 3.10.1).
	@SETC	Places a value in the condition word (see 3.10.4.1).
	@JUMP	Bypasses a portion of a runstream (see 3.10.4.3).
	@TEST	Tests the condition word when determining portions of the runstream to be processed or bypassed (see 3.10.4.2).
Checkpoint and Restart Statements	@CKPT @@CKPT	Establishes a run checkpoint dump that may be used for restart at some future time (see 11.2.1).
	@RSTRT	Restarts a run at some previously taken checkpoint (see 11.3.1).
Program Execution Statements	@processor	Executes a processor (i.e., @COB for COBOL compiler, etc.) (see 3.9).
	@XQT	Initiates the execution of a program (see 3.4.4).
	@EOF	Separates data within the runstream (see 3.4.4.3).

Table 3-1. Summary of Executive Control Statements (continued)

Statement Group	Control Statement	Description
Data Preparation Statements	@END	Terminates a data file (see Volume 3-5.2.1).
	@FILE	Causes the direct creation of a file containing data taken from the runstream (see 3.8.1).
	@ENDF	Terminates the data that follows the @FILE statement (see 3.8.2).
	@ENDX	Terminates CLIST mode (see 5.5).

3.4. SCHEDULING CONTROL STATEMENTS

3.4.1. Run Initiation (@RUN)

Purpose:

Identifies the run to the Executive and provides the information needed for accounting and scheduling purposes. The @RUN control statement must be the first statement of each run.

All parameters in the @RUN control statement are optional if user-ids are not configured in the system. If user-ids are configured, the account parameter is required. For this configuration a user-id is required for all batch runs, and for all demand runs, unless the Terminal Security System (TSS) is configured.

Format:

@RUN,priority/options run-id,acct-id/user-id,project-id,runtime/deadline,pages/cards,start time

Parameters:

priority — Indicates the preference this run should be given in relation to all other runs which are available for execution. This parameter consists of a single alphabetic character selected from the set A-Z. The nearer the selected letter is to the beginning of the alphabet, the higher the priority assigned to the run. If a priority is assigned higher than that permitted for the specified account number, the Executive alters the priority to the highest permissible level for that account. The Executive system assigns standard priority according to the account number if omitted.

options — The options are alphabetic characters that may be given in any order. The options and their meanings are given in Table 3-2. Any of these options may be overridden by processor contingency routines.

run-id — Identifies the run to the Executive. Run-id may consist of one to six characters selected from the set A-Z, 0-9. If the specified run-id duplicates a run-id already in the system, the Executive modifies the newly submitted

run–id to make it unique. When the run–id is omitted, the Executive assigns a run–id of RUN000. The numbers 000 may vary in order to establish a unique run–id. When the run–id is modified, both the original and the modified run–ids are output on the operator's console, in the master log, and in the printer listing.

acct–id

Specifies an account number. Consists of 1 to 12 characters selected from the set A–Z, 0–9, period, and dash.

At system generation, the system is configured as either restrictive or nonrestrictive. In a restrictive system, all runs using an acct–id that is not registered with the Executive are aborted. If the system is nonrestrictive, and the submitted acct–id is not registered with the Executive, the operator is notified and can accept or reject the run.

Table 3–2. @RUN Control Statement, Options

Option Character	Description
B	Treat the run input from a demand terminal as batch input and schedule accordingly.
C	Terminate the run if punched card estimate is exceeded. This option has no effect on demand runs.
E thru L	Designates the initial size of the PCT and is necessary only for real–time programs. The letters have the following meaning: E = two main storage blocks, F = three main storage blocks,..., L = nine main storage blocks. Any combination of options is allowed; the values are accumulated to the maximum size allowed by the site. This may be as much as 44 blocks.
N	Inhibit all postmortem and dynamic diagnostic dumping. This option overrides the W and the Y options.
P	Terminate the run if the page estimate is exceeded. This option has no effect on demand runs.
R	Reschedule the run in the event of a recoverable system failure. This option is ignored on all demand runs.
S	Process this run in sequence with the previous run submitted from the same input device. This run is not considered for execution until the previous run has terminated.
T	Terminate the run if the SUP usage estimate is exceeded. This option has no effect on demand runs.
W	Write a user program or LIB$ processor to DIAG$ only if the processor or program error terminates and give an automatic PMD of the error processor or program.
Y	Allow postmortem and dynamic diagnostic dumping of processors and programs in the systems absolute library file SYS$*LIB$.
YW	All user programs and LIB$ processors are written to DIAG$. In addition, an automatic PMD of error user programs and LIB$ processors is received.

user-id

Identifies the user of the account specified under acct-id. Consists of 1 to 12 characters selected from the set A-Z, 0-9, period, and dash.

At system generation, the system is configured as either user-id on or user-id off. When user-ids are not configured, the user-id parameter may be specified, but it is not used by the Executive. When user-ids are configured, and the submitted user-id is not registered with the specified acct-id, the operator is notified and can accept or reject the run.

When user-ids are required, a demand user need not include one on the @RUN statement. The system will use the one entered during logon to the system. If one is supplied on the @RUN statement, it must match the one given at logon.

Each batch run is required to have an @PASSWD card if a user-id on the @RUN card is required. A user is allowed to have several @PASSWD cards in the runstream; however, if any one of them is illegal, the run is removed with the message "RUN REMOVED DUE TO SECURITY ERROR" appearing in the run's print file. The same message appears if a single @PASSWD card presents an illegal password or if the user-id on the @RUN card is illegal. If the user-id is missing from the @RUN card, the run is removed and the message "MISSING USER-ID" appears in the run's print file. The @PASSWD image may not be placed after an @COL image if the @COL specifies a nonstandard read mode. If an @PASSWD card is placed within an @ADD file, the @PASSWD image is ignored and the message "PASSWORD STATEMENT IGNORED" is placed in the run's print file. The @PASSWD card may appear anywhere in the runstream between the @RUN and @FIN except as previously noted.

Example:

```
1.  @RUN        SAMPLE,123456/USER-ID,TSS,10,100
2.  @ASG,A      TAPEFILE,U9V,5364A
3.  @PASSWD     XYZ123
    •
    •
    •
4.  @FIN
```

project-id

Classifies the run for accounting purposes, and provides for the insertion of an implied qualifier for filenames. This parameter may consist of 1 to 12 characters from the set A-Z, 0-9, -, and $. If omitted, the Executive provides QQQ$ as the project-id.

run-time

Estimated SUP usage in minutes equivalent. The Executive can be configured to maintain a quota limit on run-time for each account. If the Executive is maintaining a quota limit and the run-time specified exceeds the quota limit, the quota limit is substituted for run-time. In addition, if run-time is not specified, the quota limit value is used. When the Executive is not maintaining quota limits, and the run-time is not specified, the system standard value is used. When the estimated run-time is exceeded, the Executive:

1. notifies the operator, allowing the operator to manually terminate the run, or

2. terminates the run, if the T option was specified in the @RUN control statement or if automatic run termination was specified at system generation.

Time is specified in minutes. See 4.9 regarding max SUPs. A leading S may be used to specify seconds.

deadline

Specifies a time (based on a 24-hour clock) by which a run must be completed. The parameter format is:

[D] *hhmm*

where:

hh specifies hours
mm specifies minutes

Leading zeros may be omitted from *hhmm*. The deadline time can be specified as either time of day or elapsed time from run submission. The operational D prefix indicates a time-of-day deadline; when the D is omitted, elapsed time deadline is indicated. Examples:

D910 - Run must be done by 9:10 A.M.
D2110 - Run must be done by 9:10 P.M.
100 - Run must be done one hour after submission
230 - Run must be done 2.5 hours after submission

The deadline parameter is ignored if the run-time parameter is not specified.

If a deadline becomes critical or if a deadline cannot be met by normal scheduling, the Executive reschedules the run (raises run priority) so that it is completed, if possible, at the specified completion time. This action, however, degrades system operation since it most likely involves suspending other runs.

At system generation, the Executive can be configured to prevent the use of deadline. In addition, the Executive can be configured to maintain quota limits for all accounts which are specifically not allowed to use deadline. If a deadline is specified when deadline is not allowed, the Executive ignores the deadline.

pages

Estimate of the number of printed pages expected as output from the run. The Executive can be configured to maintain a quota limit on pages for each account. If the Executive is maintaining a quota limit, pages may not exceed the quota limit. In addition, if the pages parameter is not specified, the quota limit value is used. When the Executive is not maintaining quota limits, and the pages parameter is not specified, the system standard value is used. If the estimate is exceeded, the Executive:

1. notifies the operator and gives the operator the option of terminating the run, or

2. terminates the run if the P option is specified, or if automatic run termination was specified at system generation.

cards

Estimate of the number of cards expected as output from the run. The Executive can be configured to maintain a quota limit on cards for each account. If the Executive is maintaining a quota limit and the cards specified exceed the quota limit, the quota limit is substituted for cards. In addition, if the cards parameter is omitted, the quota limit value is used. When the Executive is not maintaining quota limits, and the cards parameter is omitted, the system standard value is used. If the estimate is exceeded, the Executive:

1. notifies the operator and gives the operator the option of terminating the run, or

2. terminates the run if the C option is specified, or if automatic run termination was specified at system generation.

start-time

The earliest time at which a run is considered for processing. Before that time is reached, the run is placed in a hold state. The format of the parameter is the same as for deadline:

[D] *hhmm*

Once the start time has been reached, the run is released from the hold state and considered for scheduling under normal priority rules.

Other @RUN control statement parameters such as deadline and priority are not interpreted until the start time has been reached; for example, the start-time parameter is considered to be the time of run submission when considering the deadline.

3.4.1.1. Postmortem and Dynamic Dump Options

Postmortem and dynamic dumping is governed by four modes of operation which are established at the beginning of a run. To establish each of these modes, the user must specify the proper options parameter (N, Y, or W) or omit these three options. The effects of each condition are described in the following paragraphs.

- Normal Mode - No Option Specified

  When the N, Y, or W options are omitted, all programs, except processors loaded from the system library (LIB$), are dumped to the diagnostic file (DIAG$) and normal PMD action occurs. (See Volume 3-3.1 and 3.2 for information concerning normal PMD action.) @PMD control statements encountered after a processor control statement are not honored and are only printed out along with the message: PMD NOT ALLOWED. The rules governing postmortem dumping also apply to the printout for diagnostic dumping. The no-option mode should be used whenever one or more programs, other than those in the library LIB$, are being debugged. This is the normal mode for user debugging of runs, but still causes less overhead as opposed to the complete capability (Y option).

- N Option Mode

  When the N option is specified on the @RUN control statement, no program or processor is dumped to the diagnostic file upon run completion. @PMD control statements are not honored, but are printed along with the message described in the preceding paragraph. Diagnostic dumps are not printed. Specify the N option whenever the run is being executed for production purposes because the need for the overhead of saving the program for a possible @PMD control statement does not exist. The N option provides minimum overhead in the run.

- Y Option Mode

  The specification of the Y option on an @RUN control statement establishes a mode in which all programs and processors in the run are dumped to the diagnostic file. All @PMD control statements are honored, and diagnostic dumps are printed. This option should be used only when a program in the LIB$ library is being debugged.

- W Option Mode

  When the W option is specified on the @RUN control statement, only user programs and LIB$ processors that terminate in error are dumped to the diagnostic file. In addition, the PMD processor is automatically loaded to print a PMD of the error program or processor (L and P options assumed). Diagnostic dumps are printed only if the program taking the diagnostic dumps terminates in error. The W option has no meaning in demand mode.

- Y and W Option Mode

  When the Y and W options are specified on the @RUN control statement, all user programs and LIB$ processors are written to DIAG$. In addition, an automatic PMD of error user program and LIB$ processors is received.

3.4.1.2. Run Recovery (R Option)

The R option on the @RUN control statement ensures that the runstream of an open run is recovered in the event of a system failure. If the R option is specified and the associated run is open at the time of a system failure, the run is reinstated in the schedule queue during a recovery bootstrap. When selected for reopening, bit 31 is set in the run's condition word (see 3.10.4). The following actions and restrictions govern the use of the R option:

1. All queued print and punch files generated by the open run being rescheduled are removed from the print and punch queues and decataloged during the recovery bootstrap with the exception of user defined files which have been @SYMed with the U-option. These files will not be decataloged; however, they will be removed from the appropriate print or punch queue.

2. The R option is ignored on all demand runs and these runs are never rescheduled.

3. The programmer is responsible for handling facility assignments in the run such that conflicts during the restart of the run, resulting in this run being aborted, do not occur. The condition word can be examined by the run to determine whether or not the run has been restarted (see 3.10.4).

3.4.1.3. Examples of @RUN Statements

The following examples and explanations assume that the Executive is not configured to maintain quota limits.

Examples:

```
1.  @RUN          R231,03412,CAPER,10/100                    . LASTRUN
2.  @RUN,C/P      R231,03412,CAPER,,300
3.  @RUN,A        201,90431010,EXODUS1,S50,/50,D830
4.  @RUN,E/TCS    Z,A-1396/TLC,SUPER,20/230,/80
```

1. Run R231 of project CAPER is assigned standard system priority by the Executive. Expenses incurred by the run are charged to account 03412. Run-time is estimated at 10 minutes with results required within one hour of run submission. Automatic run termination does not occur if run-time, page, or card estimates are exceeded unless provided for at system generation. The Executive uses standard system card and page output estimates since these parameters are omitted. The run is scheduled by the Executive for execution according to its priority (start-time parameter omitted).

2. Run R231 of project CAPER has a C priority. Automatic run termination occurs if the page output estimate of 300 pages is exceeded (P option). Expenses incurred by the run are charged to account 03412. The Executive provides system standard entry for estimating run-time (run-time parameter omitted) and assumes that there is no deadline since this parameter is also omitted. The run is considered immediately (start-time parameter omitted) and scheduled for execution according to its priority.

3. Run 201 of project EXODUS1 has a priority of A. Run expenses are charged to account 90431010. Run-time is estimated at 50 seconds with an expected punched card output of less than 50 cards. The start-time specifies that the run not be considered for execution before 8:30 AM. Unless provided for at system generation, automatic run termination does not occur if running time or output card estimates are exceeded, since these options were not specified.

4. Run Z of project SUPER has a priority of E and is to be considered for execution only after termination of the previous run input from the same device (S option). The T and C options ensure that automatic run termination will occur if the specified running time or card output estimates are exceeded. Charges incurred by the run are charged to account A-1396. TLC identifies the user submitting run Z under account A-1396. Running time is estimated at 20 minutes with the results expected within 2.5 hours of run submission. Card output is estimated at 80 cards.

3.4.2. Run Termination (@FIN)

Purpose:

Identifies the end of a run. The @FIN control statement must appear as the last statement in all runs.

Format:

```
@FIN
```

Description:

@FIN control statements cannot be continued and must be coded exactly as shown (punched into first four columns of the card).

When the @FIN control statement is encountered, the accounting routines are called and all facilities, temporary files, and main storage areas assigned to the run are released.

The @FIN control statement is never treated as data by any processors, including the ELT or DATA processors. If an @FIN control statement is encountered in an @ADD stream, the message @FIN IN ADD FILE - IGNORED is displayed.

When the @FIN control statement is encountered, the operating system prepares a printout summary that includes initiation time, termination time, page count of printed output, and all @LOG and operator's console messages.

At a demand terminal, the @FIN control statement is normally followed either by a new logon sequence or a termination entry (@@TERM).

3.4.3. Dynamic Initiation of an Independent Run (@START)

Purpose:

Permits the user to schedule independent batch runs where the runstreams for these runs have been previously created and entered into the system. This feature allows the user to automatically schedule run execution at a time of choosing (for example, on a daily basis). Runs scheduled by this control statement must be SDF and must be cataloged as either data files or data elements. The runstream may be created by:

- the DATA processor (see Volume 3-Section 6).
- the ELT processor (see Volume 3-Section 5).
- the ED processor (see SPERRY UNIVAC 1100 Series Text Editor (ED Processor) Programmer Reference, UP-8723 (current version)).
- the CTS processor (see SPERRY UNIVAC Series 1100, Conversational Time Sharing (CTS) System Programmer Reference, UP-7940 (current version)).
- a user program.

Two formats are provided for the @START control statement. Format 1 is used when all parameters from a prestored @RUN control statement are to be used except for acct-id which will be the same as the run containing the @START. Format 2 is used when changing all or part of an @RUN control statement.

All parameters in the @START control statement are optional except name.

Format 1:

```
@label:START    name,set
```

Format 2:

```
@label:START,priority/options    name,set,run-id,acct-id,project-id,run-time/
                                 deadline,pages/cards,start-time
```

Format 1 Parameters:

name	Specifies the file or element. The file or element named must contain all the control statements required for the run. The @RUN control statement must be the first statement and the end of the file or element denotes an implied @FIN control statement.
set	Specifies an octal number to be placed in T2 of the condition word (3.10.4) of the run being scheduled.

Format 2 Parameters:

Format 2 of the @START control statement results from the integration of the basic @START and @RUN control statements. When this format is used, all parameters of the @START statement replace the corresponding parameters in the prestored @RUN control statement. Parameters existing in an @RUN control statement therefore can be modified or replaced, but not deleted, by using this format of the @START control statement. Note, however, if the S option is specified, it is ignored; i.e., not entered into the @RUN statement. Similarly, if the S option is specified on a prestored @RUN control statement, it is also ignored.

The acct-id specified in the @START control statement replaces the acct-id on the prestored @RUN control statement. If the @START control statement does not contain an acct-id parameter, the acct-id of the run containing the @START control statement is used.

If user-ids are configured, the user-id of the run containing the @START control statement is always used. If a user-id is specified on the @RUN control statement of the started run, it is ignored.

The project-id must be specified on the @START control statement in order to start from a private file. This project-id must be the same project-id as the one specified for the run which created the file.

Description:

The first image of the prestored set must be an @RUN control statement which complies with the rules detailed in 3.4.1.

The @START control statement can be used when one run by another is used to generate data for input. In fact, the generating run could build a cataloged file containing an entire run control stream and then call for it to be scheduled. As a result of this facility, the @START control statement can be used to initiate parallel processing since tasks from different runs can be executed concurrently.

Demand terminals, through the @START control statement, can initiate a batch run whose control stream has been previously entered into a data file, thus eliminating the necessity of retyping the required control statements. Any run initiated from a demand terminal using the @START control statement is scheduled as a batch run with its output going to the onsite peripherals.

The @START control statement is of particular benefit at the central site for initiating prestored utility routines and standard production runs.

The @START control statement can be issued by a user program by means of the CSF$ request (see 4.10.1.1).

Examples:

```
1.  @START      FILEA.
2.  @START      FILEA.,012
3.  @START      FILEC.ELTB,,R231,03414,,,1/50
```

1. The run found in the file FILEA is scheduled for execution by this statement. FILEA must contain all the control statements needed to initiate the run.

2. FILEA is scheduled for execution; however, the normal execution sequence of the run for this file is altered by changing T2 of the condition word for the run to 12_8.

3. The run-id (R231), acct-id (03414), and the page/card (1/50) parameters are used instead of the values found in the @RUN control statement prestored in element ELTB of the file FILEC.

3.4.4. Initiating Execution (@XQT)

Purpose:

Initiates the execution of an absolute element All parameters are optional.

Format:

@label:XQT,options eltname

Parameters:

options — Options are user specified and are presented to the program initially in a register and may be retrieved by an OPT$ request (see 4.8.1).

eltname — Specifies the absolute element to be executed. If no filename is specified, TPF$ is assumed. If no eltname parameter is specified, the most recent absolute element placed in TPF$ is assumed. When no absolute element exists in TPF$, all relocatable elements in TPF$ are collected and the resultant absolute element is executed. The newly created absolute element is given the name NAME$ and placed in TPF$.

Examples:

```
1.  @XQT
2.  @XQT          FILEA.XYZ
3.  @XQT          ABC
```

1. Execute the most recent absolute element placed in TPF$. If no absolute element is present, the Collector creates an absolute element (NAME$) from all relocatable elements in TPF$.

2. Execute absolute element XYZ in file FILEA.

3. Execute absolute element ABC in file TPF$.

3.4.4.1. Initial Execution Status

When an absolute program is loaded into main storage for execution as the result of an @XQT control statement or processor call statement, only the main segment of that program is loaded. Other program segments, if they exist, are loaded only at the request of the executing program by directly or indirectly referencing the LOAD$ request.

The program begins execution at the Collector–defined start address with one activity. This activity possesses the major register set.

Certain of the initial activity's registers are preloaded by the Executive with generally useful information. These registers and the contents are:

Register	Contents
A4	program type: 4 Demand 5 Deadline batch 6 Batch

A5 Bits 0–25 contain the options from the @XQT or processor call (A=bit 25, B=bit 24,..., Z=bit 0).
Bit 34 Print file is breakpointed (BRKPT)
Bit 33 R11–R15 have been established

R1 Date (in Fieldata DATE$ format, see 4.5.1)
Bits 35–24 Month (01 = Jan. 02 = Feb, etc.)
Bits 23–12 Day of the month (01–31)
Bits 11–0 Year (last two digits of the year)

R2 Current time (TDATE$ format, see 4.5.2)
Bits 35–30 Month
Bits 29–24 Day
Bits 23–16 Year (modulo 1964)
Bits 17–0 Time in seconds from midnight

R3 Accumulated CPU time for the run (in 200–microsecond increments)

R11/12 – System Type (two words, left–justified)

'1106I', '1106II', '1108A', '1110', '1100/10', '1100/20', '1100/40', '1100/80', '1100/80A'

R13/14 – Executive Level (two words, left–justified)

R15 – Site–id (left–justified)

3.4.4.2. Initial Program State

The initial program activity gets control with from one to four banks directly accessible. These banks are implicitly or explicitly specified during the collection process. Certain aspects of the initial setup are of interest to the programmer.

3.4.4.2.1. Overlapped Addresses

The base selection (BS) value in the PSR(s) is determined by the last address of the I–bank. Thus, if the address range of the D–bank overlaps the address range of the I–bank, part of the D–bank is inaccessible.

For programs with banks based on both PSRs (for 1110, 1100/40) or Bank Descriptor Registers (1100/80), another case of overlapped addressing may occur. The PSR select designator (D12) is initially set to zero (main PSR select). This means that if the starting address is in the overlapped address range, control is received at that address under the main PSR.

3.4.4.2.2. Lowest Bank Address

The Collector allows defining banks with first address as low as zero. If a bank with first address zero is based as an I–bank, the first 0200 locations are accessible only by instruction fetch operations. This includes jumps, execute remote instructions, and all except the final fetch of an indirect addressing sequence.

If such a bank is based as a D–bank, addresses less than 01000 are inaccessible except on the 1100/80. This is because the lowest possible base selection (BS) value is zero which forces all address references less than or equal to 0777 to the I–bank base and limits.

3.4.4.2.3. Initially-Based Common Banks

If any initially-based common bank requires control at a certain address as a requirement for access (guaranteed entry), the starting address is compared to the bank entry address. The base selection (BS) value is also checked to verify that the specified address is accessible in the correct bank. For initially-based common banks to be executed on the 1110, 1100/40/80 only, the PSR select bit (D12) is set to one (utility PSR select) if the bank is based on the utility PSR. The starting bank must be the lowest address of the bank for 1100/80 only.

There are no other restrictions on initial basing of common banks. Programs can be constructed with only common banks initially based.

3.4.4.2.4. Common Bank Access

The 1110, 1100/40 PSR and 1100/80 DR include a bit (D19) which, if zero, inhibits access (via the LBJ, LIJ or LDJ instruction) to banks in the EXEC Bank Descriptor Table. This bit will be zero if a program does not use the bank names defined in SYS$*RLIB$.CERU$ to reference the banks. The Collector includes as a part of the absolute element a list of all common banks referenced in this manner.

This rule serves two purposes. First, it aids in program debugging as accidental references may be caught immediately. Second, the system's processing is more efficient since it is known in advance which banks a program may reference.

3.4.4.3. Program Data Separation (@EOF)

Data images to be read by the program may follow the @XQT control statement. The program uses the system reference READ$ (or equivalent) in gaining access to all data images. When an executive control statement other than an @EOF is detected by READ$, further reading by the program is inhibited and an end-of-data return is given. Those data images not read by the program are bypassed when the program is finished (a message denoting this is placed in the run's print file).

Purpose:

Used as a file divider (general sentinel) within the data stream which follows the @XQT control statement (or processor control statement) that can be bypassed (read) by the user program.

Format:

```
@EOF  s
```

Description:

The optional parameter s is a 1-character, user-defined sentinel in column six of the control statement. This sentinel is passed to the requesting activity when the @EOF control statement is encountered. When the @EOF control statement is detected by READ$ request (see 5.2.1) an abnormal return is made to the requesting activity and the s parameter is placed in bits 5 - 0 of the activity's A0 register. A subsequent READ$ request causes the next image to be transmitted. The @EOF portion of the @EOF control statement is not transmitted to the user.

For additional information on using @EOF statements and detecting end-of-data see 5.2.1.

An example where the @EOF control statement is used is:

```
@XQT PROGX
. . . . . .
data of part 1
. . . . . .
@EOF A
. . . . . .
data of part 2
. . . . . .
@XQT PROGY
```

All data between the two @XQT statements are to be read by PROGX. The @EOF control statement serves as a marker between the two logical data sets.

3.4.4.4. Bank Referencing

A program may consist of up to 250 banks plus the PCT. Each activity of the program may reference two (1106, 1108, 1100/10/20) or four (1110, 1100/40, 1100/80) of the banks without performing any explicit action other than normal instruction execution. These banks are called based banks; that is, a PSR base and an SLR limits set currently describe the banks.

3.4.4.4.1. Visible Banks

An instruction can reference addresses from 0 to 0777776. A visible bank is one that is currently based and whose address range is partially or completely accessible to the instruction being executed. It is possible for a based bank to be partially or completely invisible to the current instruction. For example, if an I-bank defined as the address range 01000 to 0177777 and a D-bank defined as the address range 040000 to 0201777 are currently based on a PSR, only the D-bank addresses from 0200000 to 0201777 are visible to any instruction. Any lower address references the I-bank address.

The same effect can be achieved by different means on an 1110, 1100/40, 1100/80 System. The PSRs or DRs allow more possible combinations of banks and address ranges. If an address is visible under the same PSR or DR as the current instruction, no attempt is made to look further. If an address is not visible and bit 18 of the PSR is set to one, automatic switching of PSRs takes place (1110, 1100/40 only). Thus, the same address may have two different meanings depending under which PSR the referencing instruction is being executed. On the 1100/80, the effect is as if D18 is always set.

3.4.4.4.2. Switching Between Banks (LBJ, LIJ, LDJ)

The set of based banks available to an activity may be altered by the execution of certain instructions or by certain Executive Requests. Each bank defined during the collection process is assigned an index known as a Bank Descriptor Index (BDI). This index is the position in the Bank Descriptor Table (BDT) of information pertaining to the bank. This index may be obtained within the program by referencing the bank name as given on the IBANK or DBANK Collector directive. BDIs are also defined for some common banks and may be obtained by referencing the tags as defined in element SYS$*RLIB$.CERU$.

These BDIs are used during the execution of the LIJ, LDJ, or LBJ instruction to cause the replacement of one based bank with another bank. (The instructions are simulated on 1108, 1100/10/20 Systems). The LIJ instruction causes the replacement of the I-bank of the active PSR or DR (PSR under which the current instruction is being executed); the LDJ instruction causes the replacement of the D-bank. The LBJ instruction on the 1100/80 causes the replacement of either I or D-bank (LBJ is simulated on all other Series 1100 machines). For example:

```
LXI,U    X11,RMATH$
LIJ      X11,SIN$
```

causes the I-bank of the active PSR to be replaced by the bank whose BDI is given by the tag RMATH$. The instruction then jumps to the tag SIN$ as defined after the replacement has occurred. The BDI of the bank being replaced and the address following the LIJ instruction are captured in X11 by the LIJ instruction. A return to the next instruction with the original bank restored is achieved by:

```
LIJ      X11,0,X11.
```

LBJ (Load Bank and Jump, function code 07-17) is similar to LIJ and LDJ. With LBJ, however, the bank to be replaced is determined by bits 33 and 34 of the index word of the instruction - X_a initial. The format of X_a at a call is:

E	I/D	PSR	not used	new BD index	not used
35	34	33	32 30	29 18	17 0

where:

E = 0, user BDT
  = 1, and PSR D19=1, EXEC BDT
  = 1, and PSR D19=0, undefined sequence interrupt

I/D and PSR determine which bank is replaced.

I/D=0	PSR=0, main I-bank	BDR 0 (1100/80)
I/D=0	PSR=1, utility I-bank	BDR 1 (1100/80)
I/D=1	PSR=0, main D-bank	BDR 2 (1100/80)
I/D=1	PSR=1, utility D-bank	BDR 3 (1100/80)

new BD index index into BDT pointing to bank to be loaded

The format of X_a on the return is:

E	I/D	PSR	not used	previous BD index	reentry address
35	34	33	32 30	29 18	17 0

where:

E = copy of E bit from old BDI

I/D and PSR are unchanged from X_a initial

previous BD index — copy of old BDT index

reentry address — relative address of instruction following LBJ

Because 1106, 1108, 1100/10/20 have no utility PSR, bit 33 of X_a should never be set on LBJ instructions. If an LBJ is executed on 1106, 1108, 1100/10/20 with bit 33 set, the user is given an ERR MODE contingency type 10 code 32 (see 4.9.3). On 1106, 1108, 1100/10/20 Systems bit 33 of the return word is always clear following LIJ, LDJ, and LBJ. This reduces the danger of misinterpreting bit 33 when entering a bank by LIJ or LDJ and returning by LBJ. On the 1110, 1100/40 Systems, however, bit 33 is set or clear following LIJ and LDJ, depending on whether the R-bit of the old BDI is set or clear. For LBJ (and LIJ/LDJ on 1100/80, 1108, and 1100/10/20 Systems) bit 33 is the PSR indicator and must be set by the user before LBJ is executed. Because of these different uses of bit 33, entering banks by LIJ/LDJ and returning by LBJ can lead to errors on 1110 and 1100/40 Systems and is best avoided. The h- and i-bits of the instruction word should not be used; their use causes an illegal operation interrupt.

Once a bank is based, it remains based for the life of the activity or until some positive action is taken to replace it. When a bank is replaced, it is not necessarily physically overlayed by the bank. If the old bank is to be overlayed, its contents are automatically saved. It is then restored from the saved copy if it is called for again. Write protected (read only) banks are only saved once as, by definition, their contents cannot change.

3.4.4.4.3. Static versus Dynamic Banks

When a bank is requested, it must be loaded if it is not already in main storage. Also, if the space occupied by the bank is to become available for reuse when it is no longer based, the Executive must keep count of the number of activities for which it is based. In order to achieve this, the Bank Descriptor Table entry for the bank is marked to cause a hardware interrupt (simulated on 1106, 1108, 1100/10/20 Systems) whenever the bank is caused to be based or removed from basing by the execution of an LIJ, LDJ, or LBJ instruction. If the bank is in main storage, the interrupt serves only to allow the Executive to update counts. This interrupt does not occur on 1100/80 Systems which have hardware to update the counts.

As this operation can be quite expensive if the program is continually switching between several banks, two classes of banks are allowed. One type is the dynamic bank which is handled as described above. A dynamic bank is loaded on demand and its space may be reused as soon as it is no longer based by any activity.

The other bank type is the static bank. A static program bank is always loaded as part of the program, even if it is not based by any activity of the program. A static common bank usually resides in main storage for the time that a referencing program is in execution. The BDT entry is conditioned so that no interrupt occurs when the bank is referenced. A static bank normally takes up main storage space even when not in use but is also normally immediately available for access without requiring an

interrupt (the user should be aware that an interrupt and associated overhead still occurs on 1106, 1108, 1100/10/20 Systems due to simulation).

3.4.4.4.4. Initial Load

Options on the Collector I-bank and D-bank directives specify which banks are based as part of the initial activity created as the result of the @XQT or processor call control statement. All static banks and any dynamic banks which are based are loaded at this time. The main segment of these banks is the only data loaded. Any other portions of the banks, as well as areas for which no data is specified, are cleared to zeros or left as is depending on the presence of the B option on the @MAP control statement (see Volume 3 - 2.2.1).

As each additional dynamic bank is first referenced via an LIJ, LDJ, or LBJ instruction, it too is initially loaded. Again only the main segment is loaded and main storage clearing is based on the @MAP B option. Further references will obtain this copy along with any changes which may have been made, rather than causing a fresh copy to be loaded.

3.4.4.4.5. Active Banks

The active banks of a program consist of all static banks and any dynamic banks currently based by an activity. A program is only allowed to execute when all active banks are in main storage. Therefore, the total size of all active banks must be such that they can all fit together in user main storage.

3.4.4.4.6. Program Control Table (PCT) Referencing

The PCT is defined for each program as a write protected, static, D-bank. The program may address the PCT directly by first executing an LDJ, or LBJ instruction using PCTBD$ as the bank name. The address of the first word of the main block of the PCT is defined as RPCTA$. This address (normally 0777000) may only be used in an index register as it requires more than 16 bits. See the discussion on the PCT$ Executive Request (see 4.8.2) and the current Software Release Documentation (SRD) for more information on PCT format.

One restriction on the use of the LDJ, or LBJ instruction for PCT referencing, exists for 1108 and 1106 Systems. The simulated write protect feature may be disabled by the site when generating the system. If this is done, an attempt to base the PCT results in a table length violation interrupt. See UNIVAC 1108 Multiprocessor System Processor and Storage Programmer Reference, UP-4053 (current version), for a further discussion of the effects of negative basing.

3.4.4.4.7. Bank Unbasing

The LIJ/LDJ/LBJ instructions may be used to unbase a bank without basing another bank in its place by specifying 0 as the BDI of the bank to be based. For example:

```
LXI,U    X11,0
LDJ      X11,$+1
```

This would cause the D-bank of the active PSR to be unbased.

3.4.4.5. Program Termination

A program terminates when the last activity of the program terminates. The information saved for a PMD consists of all static banks including the PCT and any dynamic banks based by the last activity at the time of its termination.

3.5. MESSAGE CONTROL STATEMENTS

3.5.1. Displaying a Message (@MSG)

Purpose:

Used to display messages.

Only the command field is required.

Format:

```
@label:MSG,options    message
```

Parameters:

options — See Table 3-3. If the C, H, I, or S options are omitted, the S option is assumed.

message — Contains the message to be displayed. The length of this parameter is variable with a limit of 48 characters (including embedded blanks). Start-of-message is defined as the first nonblank character encountered. End-of-message is signified by the last character prior to an end of line, a comment, or 48-character limit, whichever occurs first.

When the message is displayed, it is prefixed with the run-id.

The (,) and (/) are not permitted as the first character of a message and also the continuation character (;) and the blank-period-blank sequence which introduces a comment are not permitted as part of the message.

Table 3-3. @MSG Control Statement, Options

Options Character	Description
C	Displays message on communications console devices.
H	Displays message on hardware confidence console devices.
I	Displays message on I/O console devices.
N	Suppresses message display. Message appears only in the user's print file. N option overrides W option and is used to place a comment in the print file.
S	Displays message on operator's (system) console devices.
W	Holds run until operator responds to message display. The message specified by the user is displayed and requires a response by the operator. This response may be up to 60 characters in length. A batch run may be aborted by answering X. For all demand runs where the print file has not been breakpointed (BRKPT), the response appears at the demand terminal. The response appears in the breakpointed (BRKPT) file if the print response allows the run to continue.

Description:

The message can be directed to one console category only. The order of precedence is I, C, H, and S. If none of these options is present, the message is displayed on the operator's console. Explanation of the console message categories is given in 4.6.

The W option can be used to direct the operator in loading and general management of an arbitrary device for which the loading, etc., is not taken care of automatically by the Executive.

See 4.6.1 (COM$) for the EXEC linkage which permits this function to be called from within a user program.

Examples:

```
1.  @MSG,I        EXPECT 2 REELS OF OUTPUT FOR FILE XYZ
2.  @MSG,W        IS REMOTE HOOKUP READY?                . ANSWER REQUIRED
```

1. Illustrates a 37-character message to be displayed at the I/O console (I option). The message is strictly informational and no operator response is expected.

2. This 23-character message requires a reply from the operator (W option). Since no particular console has been specified, the message is displayed on the operator's console. The program halts and does not continue until the operator replies to the message. The comment, ANSWER REQUIRED, is not displayed.

3.5.2. Inserting Information in the Master Log (@LOG)

Purpose:

Places user-specified information in the master log.

Only the command field is required.

Format:

```
@label:LOG    message
```

Parameters:

message — Contains a user-specified message to be placed in the master log. The length of this parameter is variable with a maximum limit of 132 characters including embedded blanks. The first nonblank character encountered marks the start of the message. The end of the message is signified by the last character prior to an end of line, a comment, or 132-character limit, whichever occurs first.

When encountered, the Executive extracts the message, prefixes it with the run identification, date, and time, and enters it in the master log.

The (,) and (/) are not permitted as the first character of a message. Also the continuation character (;) and the blank-period-blank sequence which introduces a comment are not permitted as part of a message.

Description:

See CSF$ (4.10.1.1) for the Executive linkage which permits this function to be called from within the user program.

Example:

```
@LOG          TRANSPORT PROBLEM NO. 128          . REVISED 5-1
```

In the example, the message consists of 26 characters (the blank preceding the period is considered part of the message) and is terminated by the comment:

```
REVISED 5-1
```

which is not transmitted to the log.

3.6. SYMBIONT DIRECTIVE STATEMENTS

The following paragraphs describe the control statements related to symbiont operations. See 2.4.3 and Section 5 for additional discussion of symbionts.

3.6.1. Print Output Heading Control (@HDG)

Purpose:

Provides a means of printing a heading on successive pages of primary printer output along with the print file's cumulative page number and the current date from a batch run or from a demand run when primary output is being directed into a file (see 3.6.2.1).

All parameters in the @HDG control statement are optional.

Format:

```
@label:HDG,options    heading  .printcontrol
```

Parameters:

Options — The options are:

- N – Suppresses printing of heading, date, and page number
- P – Starts page numbering with PAGE 1
- X – Suppresses printing of date and page number

heading — Contains the heading which is to appear within the top margin of each page (two print lines prior to the first logical print line of the page). This parameter is of variable length and is limited to a maximum of 96 characters including embedded blanks.

.printcontrol — This field consists of a period immediately followed by a print control function (see Table 5–2). This field may be repeated up to a maximum of 378 characters including the heading.

Description:

The heading starts with the second character after the options field. Leading blanks are permitted in the heading. The end of the heading text is signified by the end of the statement, a period, or the 96-character limit, whichever occurs first. The text may be followed by print control fields as described above. A period followed by a space terminates the scan of the @HDG card. Both the heading text and print control fields are optional.

If the margin at the top of a page is nonexistent or consists of less than the after heading space plus one line, the header is suppressed and not printed. Unless suppressed (N or X options), the date and page number are printed to the right of the header. Page numbering begins with the page count current to the print file unless the P option is specified, in which case page numbering begins with PAGE 1.

The @HDG control statement does not cause the printer to space to the top of the next page. The control statements:

```
@BRKPT PRINT$
```

or

```
@BRKPT PRINT$/part-name
```

terminates any current @HDG statement in effect.

See 5.4.1 (ER PRTCN$) for the EXEC linkage which permits this function to be called from within a user program.

Any number of @HDG control statements may appear in the runstream.

The continuation character (;) and the period characters are not permitted as part of the heading text.

Examples:

```
1.  @HDG,P          PYREP COMPANY - ANNUAL REPORT
2.  @HDG,X          PAYROLL REPORT - RUN-1
3.  @HDG,N
4.  @HDG,N          .M,66,3,3
```

1. The heading PYREP COMPANY - ANNUAL REPORT is printed at the top of each printed page along with the date and the page number. Page numbering begins with page number 1 (P option).

2. The specified heading appears at the top of each printed page but page numbers and date are suppressed (X option).

3. The N option suppresses further printing of the heading, date, and page numbers.

4. The M gives a new margin setting.

3.6.2. Symbiont File Breakpointing (@BRKPT)

3.6.2.1. Primary Output File Breakpoint

Purpose:

Used to partition and redirect the primary output files, PRINT$ and PUNCH$.

Format 1:

```
@label:BRKPT,option     generic-name/part-name
```

Format 2:

```
@label:BRKPT,option     generic-name,filename
```

Parameters:

All parameters in the @BRKPT control statement are optional except generic-name.

option — L - Used to provide a method of stacking multiple files on a single magnetic tape file and provide a label for each file by which it can be referenced on a subsequent @SYM command (see 3.6.3). The label is the part-name from the @BRKPT,L command(s).

generic-name — Identifies the primary symbiont output file being breakpointed: must be PRINT$ or PUNCH$.

part-name — An internal filename (see 2.6.2) identifying a new user-defined file to which subsequent primary output is to be written. Omission of part-name directs subsequent primary output to an Executive-defined file. If the L option is used, part-name is required and specifies the label to be written identifying a new part to be written on a previously breakpointed magnetic tape file. This subfield is only applicable with Format 1 of the @BRKPT statement.

filename — Specifies the user-defined internal or external file to which subsequent primary output is to be written.

Description:

The only legal punctuation mark on the Format 1 @BRKPT control statement is a slash (/). The comma and external filename punctuation are allowed on Format 2. A period following the generic-name, part-name, or filename is illegal. The @BRKPT control statement closes the previous file part. If the part is Executive-defined, it is queued for printing or punching. If the part or filename is a user-defined file, the file is closed by writing an EOF mark. In the case of a magnetic tape file, the tape is positioned such that a new file may be started.

If the part-name in Format 1 or the filename in Format 2 is not assigned to the user, the Executive does an @ASG,A of the name. If the @ASG,A is successful, the breakpoint is performed. When the user-defined file is breakpointed back to PRINT$, the file is freed (@FREE) if the Executive performed the @ASG,A function. An unsuccessful @ASG,A of the file by the Executive results in the @BRKPT not being performed and a message defining the facilities rejection being output.

A user-defined mass storage file should be used as a part-name only once, unless a means is employed to ensure that the file data is saved. Attempts to write multiple parts into such files causes overwriting of previous parts.

User-defined breakpoint files are not automatically printed. The user must use the @SYM control statement (see 3.6.3) to queue such files for printing or punching.

See 3.6.4 for examples of the use of the @BRKPT and @SYM control statements.

See 4.10.1.1 for the linkage used to invoke an @BRKPT control statement from within a user program by means of the CSF$ service request.

The @BRKPT control statement may be used from a demand terminal. However, when breakpointing PRINT$ to a user-defined file, conversational mode is lost until the PRINT$ file is redirected to Executive control (that is, the terminal) by a subsequent breakpoint.

3.6.2.2. Alternate Symbiont File Breakpoint

Purpose:

Used to close or partition alternate print, punch, and read files defined by the user (see 5.1.2).

All parameters of the @BRKPT control statement are optional except internal-filename.

Format 1:

```
@label:BRKPT,option      internal-filename
```

Format 2:

@label:BRKPT,options ,filename

Parameters:

option	E – Inhibits EOF positioning for alternate read files on magnetic tape.
internal–filename	Identifies the alternate read, print, or punch file being breakpointed.
filename	An external or internal filename identifying the alternate read, print, or punch file being breakpointed. The comma is required preceding the filename.

Description:

The discussion on primary file breakpointing (see 3.6.2.1) is generally applicable to alternate file breakpoints. The differences are:

- The alternate file is closed in that it is no longer known to the symbionts.
- In the case of input tape files, breakpoint is normally used to prematurely terminate reading of the file. In the absence of the E option, the input tape is positioned forward to the next EOF mark so that a subsequent file on the tape may be initiated as a new alternate read file. Using the E option avoids needless tape movement when the user is finished with the tape.
- Stacking of output on tape is possible, but the user must provide the part labeling by varying the internal–filename (see 3.7.5) for each alternate file written to a tape.
- Error and status codes applicable to the @BRKPT control statement can be found in Appendix C.

See 3.6.4 for examples of the use of the @BRKPT and @SYM control statements.

See 4.10.1.1 for the linkage used to invoke an @BRKPT control statement from within a user program by means of the CSF$ service request.

- The user may breakpoint up to a maximum of 63 parts.

3.6.3. Symbiont Output File Queuing (@SYM)

Purpose:

Directs the queuing of previously–created symbiont files to a user–id, a specified device, or group of devices, for printing or punching.

Also directs currently active primary output file (PRINT$/PUNCH$) to an alternative device.

Format 1:

@label:SYM,options filename,number,device,part-name-1/part-name-2/part-name-3/.../part-name-n

Format 2:

@label:SYM,options filename,number,user-id/U

Format 1 Parameters:

options

A – Specifies all files on the tape are to be printed (punched) in the order they appear on the tape; therefore, the filename given on the @SYM statement must be a tape file.

C – Directs the file to the card punch specified in the device field. If omitted, printing is implied.

D – If this option is specified and the filename is the generic name PRINT$, the current part of the PRINT$ file is deleted at the time the file is closed and no output is produced.

F – Directs all future parts of the PRINT$/PUNCH$ files to the device specified. The device specification may be changed by another @SYM,F or may be discontinued by issuing an @SYM without the F option.

U – Inhibits uncataloging of the file when processing is completed. This option is applicable only to user-defined files.

filename — Specifies the file to be processed. If filename is a user-defined file, it must be a cataloged public file. If an internal- filename is specified, the associated file must be assigned to the run. Otherwise, filename must be a generic name (PRINT$ or PUNCH$).

number — Specifies the number of copies of the file to be produced. If omitted, one copy is assumed. A maximum of 63 copies may be specified.

device — Specifies the device on which the file is to be printed or punched. This may identify a specific onsite device, a specific remote site, or a group of onsite devices (i.e., the group might be all onsite punches, or all onsite printers). Device group and remote site identifiers are defined at system generation. If omitted, the devices associated with the run initiation device are assumed.

part-names — Specifies the labels (see 3.6.2.1) of the symbiont file parts of a multifile tape to be printed or punched. If omitted, only the first part on the tape is processed. This parameter is not applicable to mass-storage files.

Format 2 Parameters:

filename — See 'filename' under parameters for Format 1.

number — See 'number' under parameters for Format 1.

user–id	any valid user–id defined within the system; if this subfield is void, the user–id of the current run is assumed
/U	as shown, indicates that queuing to a user–id is intended rather than a device. Note the slash (/) must precede the U.

Description:

In the system generation, an association of output devices to input devices is established to allow the system to direct output files created by runstream execution to the proper output device. The @SYM control statement is used to direct a standard PRINT$ or PUNCH$ file to a device or group of devices other than that specified by system generation, or to direct a user file to a device for processing. The @SYM control statement may be used to queue any SDF file.

All user–defined files processed by the @SYM control statement are uncataloged after processing unless the U option is specified. If multiple @SYM control statements are submitted for a given file, each @SYM control statement must contain the U option to ensure that the file is not uncataloged between the processing of the individual @SYM statements. More than one copy of a single file may be produced on a given device through the use of the number parameter without the U option.

When filename is PRINT$ or PUNCH$, the directive applies only to the current primary output (print or punch) part being created. In this case, the primary output cannot have been breakpointed to a user–defined file.

The order in which the part names are specified on the @SYM control statement must correspond to the order in which the parts are located on tape. However, not all parts on the tape need be processed. Any parts located on tape between those named on the @SYM control statement are bypassed.

Format 2 is used to request the queuing of symbiont files to a user–id rather than a device. As specified in the parameters section, the file may be queued either to the specified valid user–id or the user–id for the active run. If the user–id specified is not valid, no user–id is found for the run or user–ids are not configured, the message ILLEGAL USERID SPECIFIED is output.

See 4.10.1.1 (CSF$) for the linkage used to invoke the @SYM control statement from within a program.

Error and status codes applicable to the @SYM control statement can be found in Appendix C.

3.6.4. @BRKPT/@SYM Control Statement Usage

The following example illustrates the usage of the @BRKPT and @SYM control statements:

```
1.  @RUN
2.  @ASG,CP       MTAPE,T,1234
3.  @ASG,CP       SYMFILE,F
4.  @BRKPT        PRINT$/MTAPE
    •
    •
5.  @BRKPT,L      PRINT$/LABEL1
    •
    •
6.  @BRKPT        PRINT$/SYMFILE
7.  @FREE         MTAPE
```

```
8.  @SYM          MTAPE,,PR1,MTAPE/LABEL1
    •
    •
9.  @BRKPT        PRINT$
10. @FREE         SYMFILE
11. @SYM,U        SYMFILE
12. @SYM,U        SYMFILE,3,RMSITE
13. @BRKPT        PUNCH$
14. @SYM,C        PUNCH$,,RMSITE
15. @FIN
```

This run partitions its primary print output into five parts and its primary punch output into two parts.

The first and last print parts, and the first punch part, are handled automatically by the Executive, and are processed on the devices associated with the run initiation device.

The second and third print parts are stacked on the magnetic tape file MTAPE, with the labels MTAPE and LABEL1 respectively. Line 8 prints these parts on onsite printer PR1. Tape file MTAPE is then uncataloged.

The fourth print part is written on mass storage file SYMFILE. Line 11 prints this part on one of the devices associated with run initiation. Line 12 prints three copies of the same part at remote site RMSITE. SYMFILE is not uncataloged (note requirement for U option to allow multiple @SYM control statements).

Line 7 and 10 are required, since cataloging does not occur until a file is freed.

Line 14 punches the second punch part at remote site RMSITE. Note that the C option is required; otherwise, printing would occur. Also note that no other user directives (such as assignment of a file) are required to direct this part to the remote site, since it is Executive-defined.

3.6.5. Card Reader Mode Control (@COL)

(See also 3.6.6.)

Purpose:

Permits the user to switch read mode from the defined system standard to the read mode specified by the first parameter on the @COL control statement. The @COL control statement is only valid when read from an onsite card reader.

Only the sentinel parameter is optional in the @COL CB control statement.

Format:

```
@COL   CB,sentinel
```

Parameters:

sentinel — Specifies user-defined sentinel for terminating the nonstandard read mode data input stream. The sentinel may consist of from one to five characters.

The nonstandard read mode is terminated by encountering a termination control statement in the input runstream. The termination

control statement consists of the 1–to–5–character sentinel in the sentinel field of the @COL control statement preceded by the master space (@) character. If omitted, the standard system sentinel (ENDCL) is assumed. Use of the characters FIN results in termination of the input runstream in addition to the nonstandard read mode.

NOTE:

The termination sentinel followed by three blank cards must appear before the @FIN control statement. If not, the next few cards following the @FIN are read incorrectly.

Description:

Specifies the mode in which the input data following the @COL control statement is to be read from the control stream. The user must specify the characters CB to switch to column binary mode when input is from any onsite card reader. The specified input mode remains in effect until a termination sentinel is encountered or the end of the input stream is detected, at which time the standard system mode is restored.

To properly condition the input medium for handling the change of input mode, the control stream must be arranged so the the @COL control statement, and/or sentinel statement are followed by three blank cards which, in turn, are immediately followed by the data cards to be read in the new input mode. The termination control statement containing the end sentinel image for the input mode must follow the data cards. The @COL control statement and its accompanying sentinel statement are always processed at input time and in no case is action delayed until run initiation. In addition, the three blank cards which trail these two statements are eliminated from the runstream.

Examples:

```
1.  @COL        CB
2.  @COL        CB,NEWSN
```

1. The input mode is switched to accept column binary input. The standard system end sentinel (ENDCL) is assumed (sentinel omitted). The runstream associated with this statement should appear as follows:

```
@COL  CB

Three Blank Cards

Binary Data Cards

@ENDCL

Three Blank Cards

Continuation of Runstream
```

2. The user–specified terminating sentinel NEWSN is used to terminate the column binary input mode. The runstream for this example should appear as follows:

```
@COL  CB,NEWSN

Three Blank Cards
```

Binary Data Cards

@NEWSN

Three Blank Cards

Continuation of Runstream

3.6.6. Onsite C/SP,9000,0716 Card Reader Mode Control (@COL)

Purpose:

Permits the user to switch to a read/translate mode for the onsite C/SP, 9000,0716 card reader that is other than the system standard and is also not column binary. The user must switch back to the system standard read/translate mode before the @FIN control statement. If not done the next three cards following the @FIN are read incorrectly.

NOTE:

Translate options available will depend upon the hardware translate features that are installed.

Format:

@COL command

Parameters:

command — Specifies the mode to be used for the succeeding card reads (following the three blank cards) until a new @COL control statement (or an @FIN) is read. No sentinel is needed.

Description:

The @COL control statement may be used at any point in a runstream or may precede the @RUN control statement if it is necessary to establish a new input mode for a specific card deck. The commands are as follows:

Command	Card Code	Internal Code
1100FD	Fieldata	Fieldata
9000FD	EBCDIC	Fieldata
1100AS	Fieldata	ASCII
9000AS	EBCDIC	ASCII
ASCFD	ASCII	Fieldata
ASCASC	ASCII	ASCII

NOTES:

1. *Three blank cards are also needed after these @COL control statements (see 3.6.5).*
2. *Not all translate modes are always available.*
3. *The 716 via MSA or Byte channel requires an ASCII translator to use 9000FD, 9000AS, ASCFD, or ASCASC.*

4. *The 9000 card readers need the appropriate translate tables configured in SITE-SP for the translate mode requested.*

5. *C/SP card readers always support all translate modes.*

Examples:

```
@COL 9000FD
```

This command specifies that the succeeding data cards (after 3 blank cards) are in the EBCDIC card code and are to be translated to an internal code of Fieldata.

```
@COL ASCFD
```

This command specifies that the succeeding data cards (after 3 blank cards) are in the ASCII card code and are to be translated to an internal code of Fieldata.

3.7. FACILITY CONTROL STATEMENTS

3.7.1. Assigning Files and Peripheral Devices (@ASG)

The @ASG control statement is used to name a file, state its I/O facility requirements, and assign it to the requesting run, under the given external filename. If the file is cataloged, the facility requirements are known and need not be specified when assigning the file. The information pertaining to files and file-naming presented in 2.6 is a prerequisite to assigning and cataloging files.

The variety of I/O devices available makes several formats necessary for this statement. The four basic formats are:

1. Sector-formatted mass storage @ASG control statement (see 3.7.1.1).

2. Magnetic tape @ASG control statement (see 3.7.1.2).

3. Word-addressable mass storage @ASG control statement (see 3.7.1.3).

4. Absolute device @ASG control statement (see 3.7.1.3.1).

All user files must be assigned prior to being referenced for I/O operations. The assignments may occur in one of three ways:

1. by an @ASG control statement,

2. an Executive request from within a user program, or

3. an Executive request from within a part of the system itself such as a system processor.

The only instances where a file can be referenced without using an @ASG control statement are when cataloged files are being named on control statements or being named in the source language input to a processor, such as the Collector. The actual assignment is then made by the part of the system handling the given control statement, and information concerning the assignment is taken from the master file directory. If any information is needed for the file assignment other than its external name, an @ASG control statement must be used to assign the file. The @ASG control statement must also be used when the file is not a cataloged file. The user is always free to assign a file prior to referencing it on a control statement. In this case, the part of the system handling control statements detects

that the assignment has already been made. The @ASG and @FREE (see 3.7.4) control statements can be placed anywhere in the runstream. Dynamic @ASG and @FREE control statements may appear anywhere in the user program. These features allow the user to assign and free files as required, without tying up the files and facilities from the beginning of the run until its completion. However, the user might be forced to wait until the facility or file is made available when the request is for one of the following:

1. A magnetic tape unit that is being used by another run
2. An arbitrary device that is being used by another run
3. A cataloged magnetic tape file that is being used by another run
4. Exclusive use of a cataloged file which is being used by another run
5. A cataloged file which is being used exclusively by another run
6. A cataloged file rolled out by SECURE

To prevent the possible prolonged wait of a run when requesting an exclusive-use facility, and yet not force a run to specify all requirements before the first program (task) of the runstream, the Executive does:

1. not open a run for execution until all equipment requests for the @ASG control statements located before the first task in the runstream have been satisfied,
2. not start the execution of a program until all equipment requests for the @ASG control statements locate before the program call statement in the runstream have been satisfied.

On magnetic tape @FREE control statements (see 3.7.4), there is an option which releases just the file and not the physical unit. The saved physical unit is placed in the facility pool of the run and is available for reassignment at any point in this run. The unit is not returned to the Executive facility pool (made available to all runs) until it is reassigned and completely released or until run termination. The user reassigns facilities through normal means, confident that the request can be immediately honored, since the run facility pool is always referenced before the Executive facility pool. By using this option and, before the first program of the run, specifying the maximum amount of each type of magnetic tape the run requires at any one given time, the user has the ability to place @ASG control statements or dynamic control statements anywhere in the runstream or programs and be assured that the run or programs always immediately receive the facility requested.

If QUOTA Level 4 is on, enforcement of the number of removable disk units and the number of tape units is performed. If a run attempts to assign a file on removable disk or tape and the maximum number of units of the particular equipment group has already been assigned to the run, the request receives a FAC REJECT. In addition, the maximum amount of time a run may keep a removable disk or tape file assigned is monitored, and if the run exceeds this time limit the system automatically frees (@FREE) the file and displays the following message:

QUALIFIER*FILENAME @FREE DUE TO QUOTA TIME LIMIT

The message is displayed immediately to a demand user and is placed in the tail sheet for a batch run.

The files referenced by most of the @ASG control statements may be cataloged with read and write keys. At a later time, when the cataloged file is assigned to a run, the keys must be given in order to read from, write into, or delete the file. The keys may be given on the @ASG statement or on an @USE statement. The following table shows system action according to the keys specified at

cataloging and the keys given in the @ASG control statement. The entries FAC WARN and FAC REJ in the table indicate that FAC WARNING ddddddddddd and FAC REJECTED ddddddddddd messages are displayed and inserted into the PRINT$ file. See Appendix C for the meaning of these messages.

KEYS SPECIFIED AT ASSIGN

KEYS SPECIFIED AT CATALOGING	READ	WRITE	BOTH	NEITHER
READ	Read and Write Allowed	Abort Run with FAC REJ	Abort Run with FAC REJ	Only Write Allowed FAC WARN
WRITE	Abort Run with FAC REJ	Read and Write Allowed	Abort Run with FAC REJ	Only Read Allowed FAC WARN
BOTH	Read Allowed FAC WARN	Write Allowed FAC WARN	Read and Write Allowed	Neither Read nor Write Allowed FAC WARN
NEITHER	Abort Run with FAC REJ	Abort Run with FAC REJ	Abort Run with FAC REJ	Read and Write Allowed

The basic formats for the @ASG control statement are discussed in the following paragraphs. See the CSF$ request (4.10.1.1) for linkages used to call this control statement from within a user program.

3.7.1.1. Sector-Formatted File Assignment

Purpose:

Assigns sector-formatted mass storage files to a particular run.

All parameters on the @ASG control statement are optional except filename.

Format:

```
@label:ASG,options    filename,type/reserve/granule/maximum/placement,
                      pack-id-1/.../pack-id-n
```

Parameters:

options — See Table 3-4. The cataloging options (C, G, P, R, U, V, and W) are only valid during cataloged file creation. The C and U options are used to initiate cataloging action. The G, P, R, V, and W options place restrictions on the file when it becomes cataloged.

The option set A, D, K, Q, X, and Y is only valid when used with files which are currently cataloged. Use of the A option with any of the rest of the set guarantees their validity. Omitting the A option results in an attempt to find the file in the master file directory. A find assigns the file from the master file directory and honors the remainder of the set. A no-find results in a temporary file assign. Any attempt to delete a cataloged file (use of D or K option) requires the specification of the read and write keys, if there are any assigned to the file.

The B and M options control the saving and restoration of cataloged sector-formatted files in connection with checkpoint/restart. See 11.3.5.

Table 3-1. Sector-Formatted Mass Storage @ASG Control Statement, Options

Option Character	Description
	Options for Cataloging
B	Saves and reloads the file as part of any full checkpoint or restart.
C	Catalogs file if the run terminates normally. If the file is freed prior to termination, the file is cataloged at that time. If a file by this filename already exists in the Master File Directory, the run is placed in error mode.
G	Indicates that when this file is cataloged, it is to be guarded against having its read key, write key displayed or changed by a privileged run. The file is also prevented from being rolled out. (A privileged run may be initiated by the site manager for such purposes as producing a backup copy of the file on tape to use in cataloged file recovery.) This option can be used only if it is specifically allowed for this account and user.
I	Specifies that this file will be freed at the next program termination, regardless of the manner of termination.
P	Catalogs file as a public file. If omitted, file is cataloged as a private file and can be accessed only by those runs having the same project-id as the run which created the file.
R	Catalogs file as a read-only file. A file cataloged with the R option cannot be overwritten. The file can only be read or uncataloged. Any activity requesting to write in the file is placed in error mode (see 4.1.4).
U	Same as C option except that the file is cataloged at run termination (regardless of the manner of termination beyond this statement). The @FREE control statement causes cataloging prior to the termination.
V	Indicates that when this file is cataloged, its text is not unloaded to tape at any time. This option can be used only if it is specifically allowed for this account and user.
W	Catalogs file as a write-only file. The file can only be written into, and in the process, be extended.

Table 3-4. Sector-Formatted Mass Storage @ASG Control Statement, Options (continued)

Option Character	Description
	Options for Cataloged Files
A	Specifies that the file is currently cataloged and ensures that the file is not treated as a temporary file if the name cannot be found. A batch run is terminated if the name cannot be found in the Master File Directory.
D	Deletes cataloged file from the master file directory (uncatalog) if the run terminates normally, or when an @FREE control statement is encountered prior to run termination.
I	Specifies that this file will be freed at the next program termination, regardless of the manner of termination.
K	Same as D option except that the file is uncataloged at run termination, regardless of the manner of termination. An @FREE control statement uncatalogs the file prior to termination.
M	Saves and reloads the file as part of any full checkpoint or restart.
Q	Requests that this file assignment be honored, even if the system has disabled the file.
X	Specifies that this run is to have exclusive use of the file until the run has terminated or the file is released by an @FREE control statement or exclusive use is released via @FREE (see 3.7.4).
Y	Requests that this file be assigned only for the purpose of examining the master file directory. Exclusive use is overridden by this option but the file cannot be read or written when this option is used.
Z	Specifies that this control image is not to cause a hold condition. Control is returned, indicating if a hold state would have resulted from this assignment. Requests that this file is to be reloaded if it is unloaded. Note that this option has no effect on batch runs.
	Options for Temporary Files
I	Specifies that this file will be freed at the next program termination, regardless of the manner of termination.
T	Specifies that the file is temporary and allows it to have the same name as an unassigned cataloged file. The user need not be concerned if a currently cataloged file has the same name. If this option is omitted for temporary files, the Executive attempts to find the file in the master file directory. If a find is made, the assignment is made from the master file directory. If no find is made the T option is set and a temporary file is assigned.
Z	Specifies that this control image is not to cause a hold condition. Control is returned, indicating if a hold state would have resulted from the assignment. Note that this option has no effect on batch runs.

filename

Specifies the external name of the file to be assigned (see 2.6.1).

type

Specifies that the @ASG control statement is for a sector-formatted file and identifies the specific device required. If the device type specified is for a cataloged file, it is checked for compatibility. If not compatible, the control statement is rejected. Permissible entries for this parameter are:

- FCS – Sector-formatted mass storage in Unitized Channel storage (1106, 1108, 1100/10/20) or Extended Storage (1110, 1100/40)
- F4 – Sector-formatted mass storage simulated on FH-432 drum
- F17 – Sector-formatted mass storage on FH-1782 drum
- F40 – Sector-formatted mass storage on 8440 disk
- F24 – Sector-formatted mass storage on 8424 disk
- F25 – Sector-formatted mass storage on 8425 disk
- F30 – Sector-formatted mass storage on 8430 disk
- F33 – Sector-formatted mass storage on 8433 disk
- F50M – Sector-formatted mass storage on 8450 disk (movable head portion)
- F50F – Sector-formatted mass storage on 8450 disk (fixed head portion)
- F34 – Sector-formatted mass storage on 8434 disk
- F2 – Sector-formatted mass storage, FASTRAND Drum Model II
- F3 – Sector-formatted mass storage, FASTRAND Drum Model III
- F60 – Sector-formatted mass storage on 8460 disk
- F – Sector-formatted mass storage, type independent
- F14 – Sector-formatted mass storage, on 8414 disk
- F50 – Sector-formatted mass storage on 8405-00 disk
- F54 – Sector-formatted mass storage on 8405-04 disk

When space is not available for the specified device type, another type is substituted which satisfies the request. The following chart illustrates the order in which requests are satisfied. This order is from fastest to slowest device. If no space is available on a slower device, the order is reversed and faster devices are used.

Device Requested	Order of Satisfying Request
FCS (or no spec)	FCS,F4,F50,F54,F17,F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2
F50F	F50F,F50,F54,F17,F50M,F34,F33,F30,F40,F25,F14,F60,F3,F2
F4	F4,F50,F54,F17,F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2
F50	F50,F54,F17,F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2
F54	F54,F17,F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2,F50
F17	F17,F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2,F54,F50
F50M	F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2,F17,F54,F50
F34	F34,F33,F30,F40,F25,F24,F14,F60,F3,F2,F50M,F17,F54,F50
F33	F33,F30,F40,F25,F24,F14,F60,F3,F2,F50M,F17,F34,F54,F50
F30	F30,F40,F25,F24,F14,F60,F3,F2,F33,F50M,F17,F34,F54,F50
F40	F40,F25,F24,F14,F60,F3,F2,F30,F33,F50M,F17,F34,F54,F50
F25	F25,F24,F14,F60,F3,F2,F40,F30,F33,F50M,F17,F34,F54,F50
F24	F24,F14,F60,F3,F2,F25,F40,F30,F33,F50M,F17,F34,F54,F50
F14	F14,F60,F3,F2,F24,F25,F40,F30,F33,F50M,F17,F34,F54,F50
F60	F60,F3,F2,F14,F24,F25,F40,F30,F33,F50M,F17,F34,F54,F50
F3	F3,F2,F60,F14,F24,F25,F40,F30,F33,F50M,F17,F34,F54,F50
F2	F2,F3,F60,F14,F24,F25,F40,F30,F33,F50M,F17,F34,F54,F50
F	FCS,F4,F50,F54,F17,F50M,F34,F33,F30,F40,F25,F24,F14,F60,F3,F2

reserve

An integer specifying the number of granules required by the file (not to exceed 262,143). This parameter should give a reasonable estimate of the space needed to create or update the file. The value used for a file update must include those granules already in use. Files contained within the limits of the reserve are guaranteed creation without the delays involved when the Executive must find and allocate the space dynamically. Specification of a reserve aids the Executive allocation routines as the space is allocated in contiguous granules, if possible. Omission causes the Executive to dynamically allocate the granules as they are required by the file. For cataloged files, the reserve value is placed in the master file directory for future file updates. This parameter is ignored for a cataloged read-only file. However, for write-enabled files, the recorded value is overridden and replaced by the value given in this parameter.

The facility which enables a user to expand an initial reserve, following its first specification is referred to as "Static Expansion". With this

facility, an initial reserve can be expanded, both on the device which was specified on the first @ASG statement, and onto other units and/or device types.

Expansion of a file assignment across a set of units is accomplished by changing the initial reserve and the type field. Changing the initial reserve field initiates allocation if the specification is an amount greater than the current initial reserve and/or the highest granule assigned. The extent of allocation corresponds to the increment above the mass storage currently allocated to the file.

If the placement field was specified and the requested mass storage is not available, a facility request rejection occurs for an absolute subsystem (or unit) placement. However, default conditions regarding allocation exist for logical placements.

granule

Specifies granule size. It may be

TRK – One track (64 sectors)

POS – One position (64 tracks)

If omitted, TRK is assumed. If the file is currently cataloged, this parameter is ignored. The granularity is recorded in the master file directory. For most efficient use of mass storage, all program files should be allocated as TRK granularity because POS granularity creates unused space in program files (64 tracks are assigned for POS).

maximum

Specifies the maximum allowable length (in granules) of the file. Permissible values are as for the reserve parameter. When specified, this parameter overrides the system standard maximum specified at system generation. If omitted, the reserve parameter value or system standard is used, whichever is larger.

If a maximum was supplied when the file was cataloged, its value and the number of granules currently in use are recorded in the master file directory and used whenever the file is referenced. If a maximum is supplied on the referencing of an @ASG control statement, it is ignored, unless an @FREE has intervened.

This parameter is used to indicate that the run is to be terminated if the length of the file exceeds the number of granules specified. It is used primarily to ensure that a run-away-file situation does not occur during debugging. However, it may also be used to override the system standard for all files.

placement

This parameter is used to specify the placement of the file on an absolute or logical, controller or device.

Placement may be any of the following:

**cu*	where *cu* is controller name
**device*	where *device* is device name
a	where *a* is a letter, A to Z, representing logical controllers A–Z
an	where *n* is a number, 1 to 15, representing a logical device on controller *a*

The asterisk (*) flags this placement as an absolute request. If an absolute placement can not be done as specified, it is rejected. The Executive attempts to assign all files with the same logical specifications to the same physical device, and those with different logical specifications to different devices. If a logical specification cannot be honored, then the image is processed ignoring the placement field.

Initial reserves on placement files are not released when a cataloged file is freed. Placement on removable disk is not supported and such requests are rejected.

At file creation, placement information is placed in the directory to indicate the type of selection, (logical/absolute, controller/device) and the device chosen for initial allocation. The directory cell containing this information is not affected by subsequent assignment of the file.

On reassignment, an absolute placement specification is taken as a placement change, overriding the last device used for allocation.

Logical placement specifications have meaning only within the current run. If logical placement is specified on reassignment, allocation is directed to the device chosen for initial allocation.

pack–ids

Specifies the removable disk packs required for the file. Pack–ids consist of from one to six characters of the set A–Z, 0–9. The pack–ids for cataloged files are recorded in the master file directory and need not be specified on reassignments. Pack–id is applicable only to removable disks. If omitted, fixed disk is assumed, if disk equipment is requested.

Note that many jobs may specify the same set of removable disk packs for unique files.

Description:

The device type of a sector–formatted file can be changed to a new type when extending a file. To make the change, the file must be reassigned as it was previously assigned, but with a different equipment code (device type).

Example:

```
@ASG,C  FILEA,F4
(user program writes 100 tracks)

@ASG,C  FILEA,F2
(user program writes 200 tracks)

@FREE FILEA
```

In the given example, the device type was changed from F4 to F2 after 100 tracks were written. The end result was 100 tracks on type F4 (drum if available) and 200 tracks on type F2 (FASTRAND drum mass storage).

The following rules apply:

1. The file must be currently assigned to the run when the @ASG control statement with the new device type is submitted.

2. If space is not available on the new device type, allocation occurs on a slower device providing space is available.

3. The new device type is used on the first occurrence of additional space acquisition. The switch is allowed for both track (TRK) and position (POS) granularity. The user is not allowed to change granularity when specifying new device types.

4. There is no restriction on the number of times a file can be switched to a different device.

5. The maximum granules assigned are those assigned with the first @ASG statement, all following @ASG maximum granules are ignored, unless an @FREE has intervened.

Examples:

```
1.  @ASG,CR      FILEA,F/5
2.  @ASG,DA      FILEB/A2294B
3.  @ASG,T       FILEC,F/4/POS/5
4.  @ASG,XA      FILED,/6//8
```

1. If the run terminates normally or an @FREE control statement for FILEA is processed, FILEA is cataloged as a read-only file. Five tracks are assigned initially and the system-maximum size is assumed, as no maximum was specified.

2. FILEB is currently cataloged and is to be uncataloged if the run terminates normally. The key A2294B is required to read the file.

3. FILEC is a temporary file requiring four FASTRAND drum positions to be reserved initially. Termination is to occur if more than five positions are required.

4. FILED is currently cataloged and this run is to have exclusive use of the file. A reserve of six tracks is specified, and the run is to be terminated if more than eight tracks are used.

3.7.1.2. Magnetic Tape Assignment

Purpose:

Assigns a magnetic tape file to a run.

All parameters on the @ASG control statement are optional except filename.

Format:

@label:ASG,options filename,type/units/log/noise/processor/tape/format/dc,reel-1/reel-2/.../reel-n, expiration-period.

Parameters:

options

See Table 3-5.

The A, C, D, G, I, K, P, Q, R, T, U, W, Y, and Z options have the same meaning as on the sector-formatted @ASG control statement (see 3.7.1.1). The remaining options control the mode in which the file is recorded and read, and tape labeling.

In the absence of overriding mode options on 7-track tape assignments, the H and O options are assumed. For these assignments, mode options V and S are invalid.

For UNISERVO VIC and VIIIC 9-track tape assignments, the H and O options are assumed. For these assignments, mode options E, L, M, V, and S are invalid.

For UNISERVO 12/14/16/20/30 9-track assignments, the V and O options are assumed. For these 9-track tape assignments, mode options E, L, M, and S are invalid.

For UNISERVO 32/34/36 9-track assignments, the S and O options are assumed. For these tape assignments, mode options E, L, M, and H are invalid.

Table 3-5. Magnetic Tape @ASG Control Statement, Options

Option Character	Description
	Mode Options
E	Even parity. Not recommended for other than Sperry Univac-supplied software.
H	High density (800 FPI) tape (not available for UNISERVO 12/14/16 9-track if the hardware dual density feature does not exist on the unit).
L	Low density (200 FPI) tape (not available for 9-track subsystems).
M	Medium density (556 FPI) tape (not available for 9-track subsystems).
N	Specifies that a tape unit is to be assigned to the user, but no volume is assigned to the unit, and load messages to the operator are inhibited.
O	Odd Parity (assumed when type parameter specifies 9-track requirement).
S	Density mode of 6250 bpi (UNISERVO 32, 34, and 36 subsystems only).
V	Density mode of 1600 FPI (UNISERVO 12/14/16/20/30/32/34/36 9-track subsystems only).
	Options for Tape Labeling
F	Writes information on a labeled output tape to indicate that any reassign for input requires only verification of reel number. The F option may be used to bypass a forced change of the @ASG card when switching an assigned output tape to an input tape. Absence of the F option when creating a labeled output tape writes into the label blocks information which forces all subsequent assigns to use the same qualifier and filename that were used to create the tape. This causes reel number verification and confirmation that the correct file has been associated with the reel. The presence of the F option on the @ASG and in the tape's label allows the user to rewrite the tape even through it has not expired (and there are no other bans such as 'read-only' status).
J	Specifies that the reel loaded must be an unlabeled tape.

filename — The function and use of this parameter is the same as that specified for the sector-formatted @ASG control statement (see 3.7.1.1).

type — Specifies that the @ASG control statement is for a magnetic tape device and identifies the specific type of unit required. Permissible entries for this parameter are:

- T — tape unit, type independent
- C — UNISERVO VIC, and VIIIC 7-track tape units
- CB — UNISERVO VIC, VIIIC tape units with Fieldata to BCD translator

C9	UNISERVO VIC and VIIIC 9-track tape units
U	UNISERVO VIC, VIIIC, 12, 14, 16, and 30 7-track tape units
UB	UNISERVO VIIIC, and VIC tape units with Fieldata to BDC translator
U9	9-track tape unit, density independent (800, 1600, or 6250 bpi)
U9H	9-track tape unit, 800 bpi density
U9V	9-track tape unit, 1600 bpi density
U9S	9-track tape unit, 6250 bpi density
6C	UNISERVO VIC 7-track tape unit
6CB	UNISERVO VIC 7-track tape unit with Fieldata to BDC translator
8C	UNISERVO VIIIC 7-track tape unit
8CB	UNISERVO VIIIC 7-track tape unit with Fieldata to BDC translator
6C9	UNISERVO VIC 9-track tape unit
8C9	UNISERVO VIIIC 9-track tape unit
12	UNISERVO 12 7-track tape unit
14	UNISERVO 14 7-track tape unit
16	UNISERVO 16 7-track tape unit
30	UNISERVO 30 7-track tape unit
N	Specifies that a tape unit is to be assigned to the user, but no volume is assigned to the unit, and load messages to the operator are inhibited.
12N	UNISERVO 12 9-track tape unit
14N	UNISERVO 14 9-track tape unit
16N	UNISERVO 16 9-track tape unit
20N	UNISERVO 20 9-track tape unit
32N	UNISERVO 32 9-track tape unit
34N	UNISERVO 34 9-track tape unit

36N UNISERVO 36 9-track tape unit

12D UNISERVO 12 dual density 9-track tape unit

14D UNISERVO 14 dual density 9-track tape unit

16D UNISERVO 16 dual density 9-track tape unit

30D UNISERVO 30 dual density 9-track tape unit

The following selection sequences supplied by the Executive are:

Type	Order of Satisfying Request
T	36N, 34N, 32N, 30D, 20N, 16N, 16D, 14N, 14D, 12N, 12D, 8C9, 6C9, 30, 16, 14, 8C, 8CB, 12, 6C, 6CB
C	8C, 8CB, 6C, 6CB
CB	8CB,6CB
C9	8C9,6C9
U	30, 16, 14, 12, 8C, 8CB, 6C, 6CB
UB	8CB, 6CB
U9	36N, 34N, 32N, 30D, 20N, 16N, 16D, 14N, 14D, 12N, 12D, 8C9, 6C9
U9S	36N, 34N, 32N
U9V	36N, 34N, 32N, 30D, 20N, 16N, 16D, 14N, 14D, 12N, 12D
U9H	30D, 16D, 14D, 12D, 8C9, 6C9
8C	8C, 8CB
6C	6C, 6CB
12N	12N, 12D
14N	14N, 14D
16N	16N, 16D

The high density, 6250 bpi, may be selected explicitly by a user through the specification of the S option on an @ASG statement, or by using the assignment mnemonic 'U9S'. In addition, tapes are assigned with 6250 bpi density if the 'T' or 'U9' assignment mnemonics are used, no density options are specified, and the selected tape unit is a U32, U34, or U36.

The following magnetic tape assignments do not have a second choice:

30,30D,32N,34N,36N,20N,12,14,16,12D,14D,16D,8CB,6CB, 8C9,6C9.

The use of type T or U9 is encouraged as it gives the system more freedom in assigning units. When using type T, only those functions and options compatible with all types of units may be specified.

units

Specifies the number of tape units required, and may be integers 1 or 2. If omitted or a number other than 1 or 2 is specified, the Executive assumes that one unit is required. The number of units assigned is not retained in the master file directory upon cataloging. See Description for absolute assignment.

log

Assigns a single letter indicating a logical channel. The Executive attempts to assign all files with the same letter to the same physical channel and those with different letters to different channels. The letter specified is not placed in the master file directory upon cataloging. This parameter permits more efficient I/O operation because of the separate channels.

noise

Specifies an integer from 0 to 99 which overrides the standard system noise constant. If omitted, the standard system noise constant is assumed (see 6.4.2.1).

processor/tape

Specifies the type of translation required. The processor subfield defines the format of the data in the processor, and the tape subfield defines the format of the data on tape. The supported processor/tape mnemonics are:

Processor Code	Tape Code
ASCII	BCD
ASCII	EBCDIC
ASCII	FLDATA
ASCII	XS-3
EBCDIC	BCD
FLDATA	ASCII
FLDATA	BCD
FLDATA	EBCDIC
FLDATA	XS-3
XS-3	ASCII
XS-3	BCD
XS-3	EBCDIC

Additional processor/tape mnemonics can be defined by a site. Translation is turned off by specifying equal processor/tape mnemonics.

format

For byte devices only (12, 14, 16, 20, 30, 32, 34, 36). Specifies the data transfer format for the word-to-byte conversion. The data transfer mnemonics are:

- Q – quarter-word
- 6 – 6-bit packed (default on 7-track)
- 8 – 8-bit packed (default on 9-track)

dc

For byte type 7-track assignments only. The allowable mnemonics are:

- DC – three 8-bit bytes converted to or from four 6-bit tape characters
- OFF – turns off data converter if assign is made from the master file directory and the file was cataloged with data conversion.

reels

Specifies the identifier for each tape reel required. Each reel identifier is limited to six characters from the set A-Z and 0-9. A maximum of 2047 reels can be specified. Reels are used and cataloged in order specified. If the file is to be cataloged, all reel identifiers are recorded in the master file directory. If omitted, the Executive directs the operator to mount blank reels on the appropriate tape units and to provide reel identifiers for each reel. If additional tape reels are requested by a TSWAP$ request (see 7.2.9), operator is requested to load the required blank reels and provide identifiers for each reel.

For currently cataloged files, the reel parameter is normally void, indicating that the reels listed in the master file directory are to be used in the order in which they were created. If reel numbers are supplied, they must be of the set listed in the master file directory, but may be a subset and listed in any order, allowing the user to omit or access them in any order. If an invalid reel number is supplied, the @ASG control statement is rejected. In either case, when the known reels are exhausted and additional reels are requested, blank reels are used and their numbers added to the master file directory. (This is not allowed if the file is cataloged in the read-only state.)

expiration-period

Specifies the number of days that this file is to be retained. The maximum number allowed is 4,095 days.

The EXEC guarantees that the expiration period on all files beyond the first file are less than or equal to the first file. If on a subsequent @ASG that has an expiration period greater than that remaining on the first file, the EXEC automatically changes the expiration period of the new added file to the remaining expiration period of the first file.

Description:

The Executive normally controls the selection of units for the assignment of tape files. However, the user may direct the tape assignment to a particular control unit or device by specifying the absolute control unit or device. Absolute control unit or device specification is not the recommended procedure for assigning tapes. However, maintenance routines, real-time programs, or special hardware requirements may dictate the need for such specifications.

A device that has been placed in the reserved state by a RV unsolicited console keyin can only be assigned by specifying the absolute control unit or device on the assign statement. However, it is not necessary to have a device in a reserve state in order to assign it absolutely.

For absolute tape assignment, all parameters except type, units, and log retain their meaning as described earlier. The contents of these three parameters are:

type	Contains the name of a control unit or the name of a device preceded by an asterisk (*).
units	If the type subfield contained the name of a control unit, this subfield has the same meaning as described earlier. If the type subfield contained the name of a device, this subfield may contain the name of a second device preceded by an asterisk.
log	Must not be used. Use of this subfield causes a reject.

Examples:

```
1.  @ASG,A        FILEX
2.  @ASG,T        FILEA,T///36
3.  @ASG,TEL      FILEB,6C/2,N432
4.  @ASG,CR       FILEC,8C9
5.  @ASG,DA       FILED/4B96,8C//A,N212
6.  @ASG,U        FILEE/491671/RA1234,8C/2,707/708/709/710
7.  @ASG,T        FILEF,T,SCRTCH
8.  @ASG,T        FILEY,*CUA/2
9.  @ASG,T        FILEY,*TAPE-1,29416
10. @ASG,T        FILEY,*TAPE-2/*TAPE-3
```

1. FILEX is cataloged and all necessary options, facility requirements, and reel numbers are taken from the master file directory. The project-id of the current run is used as a qualifier.

2. FILEA is a temporary file requiring one unit selected by the Executive; one or more blank reels are used. The noise constant is to be set to 36 characters.

3. FILEB is a temporary file requiring two UNISERVO VIC tape units. It is recorded in even parity and low density. Reel number N432 is specified.

4. FILEC is to be cataloged in the read-only mode if the run terminates normally. One UNISERVO VIIIC tape unit with 9-channel capabilities is required.

5. FILED is currently cataloged but is to be uncataloged if the run terminates normally. A key of 4B96 is required to read this file. The UNISERVO VIIIC tape unit is to be on logical channel A and reel N212 is to be used.

6. FILEE is to be cataloged. It requires two UNISERVO VIIIC tape units on any channel. Reels 707 through 710 are to be used. The file is locked by the specified read (491671) and write (RA1234) keys.

7. FILEF is a temporary file and the symbol SCRTCH is used as a reel number.

8. FILEY is assigned two units of the system's choosing from the control unit named CUA.

9. FILEY is assigned to device TAPE-1; reel 29416 is to be used.

10. FILEY is assigned to devices TAPE-2 and TAPE-3.

3.7.1.3. Word-Addressable Mass Storage Assignment

3.7.1.3.1. Normal Assignment

Purpose:

Assigns word-addressable mass storage and simulated word-addressable mass storage to a particular run.

All parameters are optional on the @ASG control statement except filename.

Format:

```
@label:ASG,options    filename,type/reserve/granule/maximum,pack-id-1/
                      pack-id-2/...pack-id-n
```

Parameters:

With the exception of the following differences, the parameters of this statement are basically the same as those for the sector-formatted mass storage @ASG control statement (see 3.7.1.1).

options — Same as 3.7.1.1, except no distinction is made between file types except in the conversion of logical to physical addresses.

filename — Specifies the external name of the file to be assigned.

type — Specifies that the @ASG control statement applies to word-addressable drum format and names the specific type of recording equipment to be used. Permissible parameters are:

- D - Word-addressable storage type independent
- D4 - Word-addressable storage FH-432 drum
- D17 - Word-addressable storage FH-1782 drum
- DCS - Word-addressable storage, Unitized Channel Storage (1108, 1100/10/20), or Extended Storage (1110, 1100/40)
- D50 - Word-addressable storage on 8405-00 disk
- D54 - Word-addressable storage on 8405-04 disk
- D14 - Word-addressable storage on 8414 disk
- D24 - Word-addressable storage on 8424 disk
- D25 - Word-addressable storage on 8425 disk
- D40 - Word-addressable storage on 8440 disk
- D30 - Word-addressable storage on 8430 disk

D33 – Word–addressable storage on 8433 disk

D34 – Word–addressable storage on 8434 disk

D50M – Word–addressable storage on 8450 disk (movable head portion)

D50F – Word–addressable storage on 8450 disk (fixed head portion)

Use of the D entry is recommended, since it allows the Executive freedom in allocating file space.

The order of satisfying equipment requests is the same as for sector–formatted (see 3.7.1.1).

reserve — Same as 3.7.1.1, except the entry is in number of words (1792 wds/trk).

granule — Same as 3.7.1.1.

maximum — Same as 3.7.1.1, except the entry is in number of words (1792 wds/trk).

pack–ids — Same as 3.7.1.1.

Description:

Word–addressable files cannot be used as program files. The mass storage allocation routine attempts to satisfy word–addressable requests in the same manner as it satisfies sector–formatted requests.

Examples:

```
1.  @ASG,CR      FILEA,D4/275/TRK
2.  @ASG,T       FILEB,D/2217/TRK
3.  @ASG,UP      FILEC,D17/17560/TRK
```

1. FILEA is to be cataloged in the read–only mode on the FH–432 drum upon normal termination of the run or when an @FREE control statement (see 3.7.4) is encountered. Initial length of FILEA is one track. The standard system maximum is assumed for maximum subfield.

2. FILEB is a temporary file requiring an initial reserve of two tracks and residing on mass storage selected by the Executive.

3. FILEC is to be cataloged on the FH–1782 drum as a public file at run termination (regardless of the manner of termination), or when the @FREE control statement is encountered. Initial length of the file is 10 tracks. Standard system maximum is assumed.

3.7.1.3.2. Absolute Device Assignment

Purpose:

Used primarily for the assignment of special I/O devices, communications equipment and symbiont controlled devices, and online maintenance of all system peripherals. If the device is configured as a symbiont controlled device, the device must be placed in the reserved state by the operator prior to the assignment.

All parameters on the @ASG control statement are required, except label and pack-id. For assignment of communications devices through a C/SP, see C/SP Operating System Series 1100 Supplement, UP-7917 (current version).

Format:

@label:ASG,options filename,type,pack-id

Parameters:

options — Same as 3.7.1.1.

filename — Same as 3.7.1.1.

type — Specifies:

1. Absolute control unit; the Executive selects the specific device if more than one unit exists
2. Absolute device name

For absolute control unit assignment, the type parameter contains the name of the control unit preceded by an asterisk (*).

For absolute device assignment, the type parameter contains the name of the device preceded by an asterisk (*).

If a communications device is assigned, the type parameter contains the name of the line to be assigned. An asterisk does not precede the name of a communications device.

A disk can be assigned for use with the arbitrary device handler by using the absolute device assignment. The format is:

```
@ASG,options  filename,*device,pack-id
```

The requested unit must be in the reserved state to satisfy this type request (units can only be put in the reserved state by the operator). The pack-id parameter is used in a load message to instruct the operator to mount a specific pack on a specific device.

```
1.  @ASG        FILE,*PRINTR
2.  @ASG        FILE,*DISC1,QXYZ
```

1. The device called PRINTR is assigned to the run.

2. A disk named DISC1 is assigned to the run. The operator is requested to mount pack QXYZ on DISC1.

3.7.2. Tape Unit Mode Control (@MODE)

Purpose:

Changes the mode settings initially set by a previous tape @ASG control statement. The file must be currently assigned to the run.

All parameters on the @MODE control statement are optional except filename.

Format:

```
@label:MODE,options    filename,noise/processor/tape/format/dc
```

Parameters:

options	Same as mode options in Table 3–5.
filename	Specifies external name of tape file to which mode change applies.
noise	Same as 3.7.1.2.
processor	Same as 3.7.1.2.
tape	Same as 3.7.1.2.
format	Same as 3.7.1.2.
dc	Same as 3.7.1.2.

Description:

With the @MODE control statement, options (modes) are never assumed in the absence of others. The specified options are not placed in the Master File Directory (MFD), since they apply only to the current assignment.

See the CSF$ request (4.10.1.1) for the linkage used to call this control statement from within a user program.

Example:

```
@MODE,EL        FILEY,30
```

The initial mode settings assigned to FILEY are overridden by the E, and L options specified in the @MODE control statement. A noise level of 30 is also specified for FILEY.

3.7.3. Independent Cataloging of Files (@CAT)

Purpose:

The @CAT control statement is used to catalog one or more files without having them assigned to the run, such as when building the initial master file directory or when a previously prepared tape file is to be cataloged.

Two formats are provided for @CAT control statement. Format 1 is used for cataloging tape files and Format 2 is used for cataloging mass storage files.

All parameters on the @CAT control statement are optional except filename.

Format 1:

@label:CAT,options	filename,type///noise/processor/tape/format/dc,reel-1/reel-2/.../reel-n, expiration-period

Format 1 Parameters:

options	See Table 3-6.
filename	Specifies the external name of the file to be cataloged.
type	Same as 3.7.1.2. If omitted, the Executive assumes that the request is for a sector-formatted file.
noise	Same as 3.7.1.2.
processor	Same as 3.7.1.2.
tape	Same as 3.7.1.2.
format	Same as 3.7.1.2.
dc	Same as 3.7.1.2.
reels	Same as 3.7.1.2.
expiration-period	Same as 3.7.1.2.

Table 3-6. @CAT Control Statement, Options

Option Character	Description
E	Even parity.
G	Indicates that when this file is cataloged, it is to be guarded against having its read key, write key, and similar protection overridden by privileged runs. (A privileged run may be initiated by the site manager for such purposes as producing a backup copy of the file on tape to be used in cataloged file recovery.) This option can be used only if it is specifically allowed for this account and user-id.
H	High density.
J	Unlabeled tape.
L	Low density.
M	Medium density.
O	Odd Parity.
P	Catalog file as public file.
R	Place in read-only state.
S	Density mode of 6250 BPI (UNISERVO 32, 34, and 36 subsystems only).
V	Indicates for mass storage that when this file is cataloged, its text is not to be unloaded to tape at any time. For tape, indicates cataloged density is to be 1600 FPI. This option for mass storage can be used only if it is specifically allowed for this account and user-id.
W	Place in write-only state.

Format 2:

@label:CAT,options filename,type/reserve/granule/maximum,pack-id-1/pack-id-2/.../pack-id-n

Format 2 Parameters:

options — The G,P,R,V, and W options are the only valid options (see Table 3-6).

filename — Same as 3.7.1.1 and 3.7.1.3.

type — Same as 3.7.1.1 or 3.7.1.3 depending on file format. If omitted, the standard system entry (F) is assumed.

reserve — Same as 3.7.1.1 and 3.7.1.3.1.

granule — Same as 3.7.1.1 and 3.7.1.3.1.

maximum — Same as 3.7.1.1 and 3.7.1.3.1.

pack-ids — Same as 3.7.1.1 and 3.7.1.3.1.

See the CSF$ request (4.10.1.1) for the linkage used to call this control statement from within a user program.

Examples:

```
1.  @CAT,P        FILEY//WLOCK,T,N247/N248
2.  @CAT,W        FILEX//WLOCK,F/7/TRK/10
3.  @CAT,R        WADDRUM,DCS/1700/TRK/2000
```

1. FILEY is to be cataloged as a public file with the write-key WLOCK. The device type is specified as tape with the specific tape device selection made by the Executive. The file is recorded on the two reels identified as N247 and N248.

2. File FILEX is to be cataloged in the write-only mode. The key WLOCK is required to write in the file. The recording device is specified as sector-formatted mass storage with the specific device selected by the Executive. At assign time, seven sector-formatted mass storage tracks are initially reserved for the file. The run is placed in error mode if more than 10 tracks are used.

3. File WADDRUM is to be cataloged in the read-only state. The recording device is specified as word-addressable format, unitized channel storage. One track is reserved for the file at assign time and the run is placed in error mode if the file exceeds two tracks.

3.7.4. Releasing Files and Peripheral Devices (@FREE)

Purpose:

Unassigns files and releases their facilities, reels, and exclusive use areas assigned to the run. Files and facilities should be released the moment they are not needed. If they are not released by means of an @FREE control statement, they are retained until run termination.

All parameters on the @FREE control statement are optional except filename.

Format:

```
@label:FREE,options      filename
```

Parameters:

options — See Table 3-7.

filename — Specifies the internal or external name of the file to be deassigned. This entry must agree with the filename specified by the @ASG control statement (see 3.7.1) or equated to the file by the @USE control statement (see 3.7.5).

Table 3-7. @FREE Control Statement, Options

Option Character	Description
A	Releases only the specified internal name relationship to the file.
B	Releases only the specified internal name associated with the file unless it is the only internal name attachment, in which case the entire file is freed.
D	Uncatalogs a cataloged file. The file must have been assigned with the correct read/write keys.
I	Inhibits final cataloging action if the file was assigned by an @ASG control statement with a C or U option.
R	Releases the file assigned but retains the internal name relationships to the filename, the F-cycle, and any keys specified.
S	Frees the file but retains the physical tape unit.
X	Releases exclusive use of the file. The file, however, is not deassigned from the run.

Description:

Combinations of @FREE options have the following effect:

BD Uncatalogs the file if the specified internal name is the only internal name attached to the external name. Otherwise, the file remains cataloged.

BI Inhibits final cataloging action if the specified internal name is the only internal name attached to the external file name.

AR Releases the file assigned and the specified internal name relationship to the filename if the specified internal name is the only internal name attached to the external name. Otherwise, the file is released and other internal name(s) relationships to the file name are retained.

A file that is deassigned by an @FREE control statement can no longer be referenced by the run. It can, of course, be reestablished by an @ASG control statement provided its facility requirements can be met.

The actions taken by the system when a file is deassigned by an @FREE statement (and the S option was not specified) are discussed below.

For a temporary file (not cataloged or to be cataloged):

Mass storage – The mass storage area is made available as file space for other runs.

Tape – Units are released for use by other runs.

Other equipment (communications, etc.) – The device is released for use by other runs.

For a file being cataloged (C or U option on @ASG control statement):

Mass storage – Catalog entry is made in the master file directory, and the mass storage area containing the file is held. The file can now be referenced by other runs.

Tape – Catalog entry containing reel numbers is made; units are released for other runs.

For a file being uncataloged (D or K option on @ASG control statement):

Mass storage – Same as for a temporary file except that the file area is not released until all runs currently using the same file are also finished. It is no longer available for assignment.

Disk – Same as sector-formatted mass storage.

Tape – Units are released for use by other runs. The file is no longer available for assignment.

See the CSF$ request (4.10.1.1) for the linkage used to call this control statement from within a user program.

Examples:

```
@ASG,C          FILEA,F/5
@ASG,T          FILEB,F/4/POS/5
@ASG,X          FILEC,/6//8
@ASG,CR         FILEX,8C//A,R121
@ASG,T          FILEY,8C//B,R212
@ASG,D          FILEZ,8C//C,R111


1.  @FREE,I     FILEA
2.  @FREE       FILEB
3.  @FREE,X     FILEC
4.  @FREE,S     FILEX
5.  @FREE,R     FILEY
6.  @FREE       FILEZ
```

1. FILEA is to be cataloged upon normal run termination or by execution of the @FREE control statement as specified by the C option on the @ASG control statement. The @FREE control statement, however, inhibits cataloging of FILEA because of the I option.

2. FILEB is deassigned and all filenames are released (with no special considerations).

3. Exclusive use of FILEC (currently cataloged sector-formatted file designated as exclusive use for current run) is released. The exclusive use obtained by the @ASG control statement is released by the X option on the @FREE control statement.

4. FILEX (designated to be cataloged at normal termination of the run or by execution of the @FREE control statement) is deassigned. The UNISERVO VIIIC tape unit is retained for run use (S option). The reel number (R121) is recorded in the master file directory.

5. Temporary tape file FILEY is deassigned. The internal filename relationship to the file is retained (R option specified). The F-cycle is also retained.

6. FILEZ is uncataloged from the master file directory. The UNISERVO VIIIC tape unit is released for use by other runs.

3.7.5. Attaching Internal Filenames (@USE)

Purpose:

Equates filenames so that any particular file can be referenced by more than one filename. This need arises when:

1. run construction can be simplified by using a shorter internal filename in place of a long external filename

2. identical filenames must be differentiated

3. internally programmed filenames must be connected to external filenames

The information presented in 2.6.2 on file naming is a prerequisite for understanding internal and external filename relationships.

The @USE control statement has two formats: equating internal filenames to external filenames (Format 1), and equating internal filenames to internal filenames (Format 2).

All parameters are *required* except label.

Format 1:

@label:USE,options	internal–filename,external–filename

Format 2:

@label:USE,options	internal–filename,internal–filename

Parameters:

options	The I option is the only recognized option. Any other options used will be ignored. It is very similar to the I option associated with the @ASG control statement (see Table 3–4). It specifies that the internal filename and associated external file will be freed upon the next program termination (normal or abnormal).
internal–filename	Specifies the name by which the file can be referenced within the run after the @USE control statement is encountered in the control stream.
external–filename	Specifies the full external name of the file. The external name usually takes the form qualifier*filename (F–cycle)/read–key/write–key/. The exception is when external name has been previously used in an earlier @USE command.

Description:

All internal filenames equated to an external filename are listed and maintained for the run. Once equated, the user can reference the file by its internal or external filename from within a program or the runstream. If a conflict of filenames exists, it is the user's responsibility to attach an internal name to the file (with the conflicting external name before any references to that file are attempted). The internal filename list is always searched first on file reference.

If a no-find condition occurs on the internal names, the external filename list is searched.

Multiple internal filenames can be attached to an external filename.

See the CSF$ request (4.10.1.1) for the linkage used to call this control statement from within a user program.

```
1.  @USE        FILEB,COMPANY*PAYROLL
2.  @USE        COST,FILEB
3.  @USE,I      A,FILEA
```

1. The internal filename FILEB is equated to external filename COMPANY*PAYROLL. The file can now be referenced for I/O by either the name FILEB or the name PAYROLL.

2. The internal filename COST is now a third association to the file COMPANY*PAYROLL for I/O.

3. The internal filename A is equated to external filename FILEA. The file can now be referenced for I/O by either the name FILEA or the name A. Upon the next program termination (normal or abnormal), A will be freed. FILEA will also be freed.

3.7.6. Specifying Filename Qualifier (@QUAL)

Purpose:

Specifies a standard filename qualifier for implied usage on succeeding control statements involving filenames.

All parameters in the @QUAL control statement are required except label.

Format:

```
@label:QUAL  qualifier
```

Parameters:

qualifier	Specifies name extension used to qualify subsequent filenames which are immediately preceded by an asterisk (*).

Description:

Any number of @QUAL control statements can appear throughout the control stream. Each time an @QUAL control statement is encountered, the new qualifier overrides the qualifier specified in the previous @QUAL control statement.

See the CSF$ request (4.10.1.1) for the linkage used to call this control statement from within a user program.

Examples:

```
1.  @QUAL       1STQUAL
    @FOR        *FILEA.DO/ABC
2.  @QUAL       2NDQUAL
    @FREE       *FILEA
```

1. The @QUAL control statement provided in this example specifies that all subsequent filename references, which have a preceding asterisk, be interpreted as having the qualifier 1STQUAL. For example, the @FOR control statement element shown in the example is interpreted as:

   @FOR 1STQUAL*FILEA.DO/ABC

2. The @QUAL control statement supersedes the @QUAL control statement in example 1. Therefore, subsequent filename references which have a preceding asterisk have the implied qualifier 2NDQUAL. For example, the @FREE control statement is interpreted as:

   @FREE 2NDQUAL*FILEA

3.8. DATA PREPARATION CONTROL STATEMENTS

3.8.1. Direct Creation of Card Image Files (@FILE)

Purpose:

The @FILE control statement creates SDF mass storage or magnetic tape files while the input symbiont is reading the runstream. Creating a file in this manner, rather than by means of the ELT or DATA processors, reduces overhead since it is accomplished at input time rather than at execution time. The data is handled only once in creating the file, rather than being read again from auxiliary storage with the runstream and then transferred to the file by the DATA or ELT processors at execution time. The file into which the images are placed may be either a sector-formatted or magnetic tape file. Sector-formatted files may not be temporary files except for removable disks. The @FILE statement is valid only within an @RUN, and cannot have been initiated via @START.

Format:

For each storage device, the format of the @FILE control statement is identical to the @ASG control statement for that device (except that the label field is not used).

If the device type specifies sector-formatted mass storage, the format is:

@FILE,options filename,type/reserve/granule/maximum,pack-id-1/ pack-id-2/...pack-id-n

NOTE:

The pack-ids are only used if the type field specifies removable disk.

If the device type is magnetic tape, the format is:

@FILE,options filename,type/units/log/noise/processor/tape/
format/dc,reel-1/reel-2/.../reel-n, expiration-period

Parameters:

options — The options are interpreted exactly as they would be for the @ASG statement of the appropriate device type.

Description:

For a more complete explanation of these options and the remainder of the control statement, see the @ASG statement for sector-formatted mass storage or magnetic tape (see 3.7.1.1 and 3.7.1.2).

The qualifier must either be explicitly given or the qualifier QQQ$ is assumed. For private files, the project-id is picked up from the @RUN control statement.

@FILE control statement processing is terminated upon encountering an @ENDF control statement, an @FIN control statement, or another @FILE control statement. Data images and all control statements except @COL and its accompanying end sentinel are placed into the created file (the @COL control statement and sentinel are processed immediately, and the file is marked when the mode switch is made).

@FILE control statements cannot be nested unless they use the same filename and are a tape file. Nested mass storage files will cause @FILE mode to terminate with an error message. Nested tape files with the same name will cause an EOF to be written on the tape and a new file to follow the EOF.

No run is opened until an @FIN control statement is encountered. Thus, all @FILE files will have been created.

The @FILE statement is not available from demand runs.

The C or U option causes cataloging to occur upon successful completion of the @FILE (@ENDF or @FIN encountered), rather than upon successful run termination.

3.8.2. Terminating the File Mode (@ENDF)

Purpose:

Marks the end of the images for a file created by the @FILE control statement.

Format:

```
@ENDF
```

Description:

@ENDF control statements cannot have labels and cannot be continued.

3.8.3. Direct Creation of Data Files from NTR Sites

Purpose:

An SDF-formatted mass storage file can be created directly by initiating a data-transfer device or initiating a non-data-transfer device in alternate mode.

Description:

This data-transfer file consists of images which are not sensitive to content of images (e.g., @RUN,@COL), to control images (e.g., control directions for printers, punches), to type of symbiont device (e.g., card reader, printer, punch, paper tape, disk, magnetic tape), or to image length (up to 510 words). Upon initiation of the input data-transfer device, the Executive generates an @RUN and an @FILE control statement. The public, read-only mass storage file is named QQQ$*symbiont name. The 1100 user program may use the standard system library routines of SDFI/SDFO to access the mass storage file. After an output data-transfer file is created, the user does an @SYM,C of the filename to the desired output data-transfer device or device group. If another input data-transfer file is sent, the F-cycle number is updated for the QQQ$*symbiont name associated with the device.

3.9. PROCESSOR CONTROL STATEMENTS

Two separate library files are available during the processing of a user run: the absolute library file (SYS$*LIB$) and the relocatable library file (SYS$*RLIB$).

The absolute library file (LIB$) contains the absolute elements of each standard processor included in the operating system. LIB$ may contain any other processor and executable program added by the installation.

The relocatable library file (RLIB$) contains the system-supplied relocatable elements and procedure elements which may be needed to assemble, compile, or collect the user program.

A temporary program file (TPF$) is created by the Executive for each run that is initiated. The qualifier for the filename is taken from the project-id field of the @RUN control statement. The file may be used as a scratch file for the user program's symbolic, relocatable, and absolute elements.

Processors form a special class of absolute elements which provide standard services for the user. The principal distinction is in regard to the manner in which the absolute element is invoked and the means by which information is made available to the processor.

The general format of the processor control statement is:

```
@label:processor,options    param-1,param-2,param-3,...,param-n
```

The label field is as described in 3.2.1. The processor field is the name and file location (see 3.2.2) of the absolute element desired. The following is an example of a generalized processor control statement where the processor is located in a user-specified file rather than in the system library file LIB$:

```
1.  @USER*FILE.PROG/ABS,P        FILE IN,ELTOUT,FILE OUT
```

The rules for locating the element in the processor field are slightly different from the standard rules for locating an element specified on an @XQT control statement (see 3.4.4):

1. If a filename is specified, then that file is searched for the absolute element.

2. If a filename is not specified but there is a leading period, then TPF$ is searched for the element.

3. If a filename is not specified and there is no leading period, then the system library file SYS$*LIB$ is searched for the element; if there is no find, then TPF$ is searched. The abbreviations for the standard processors (COB for COBOL, FOR for FORTRAN, etc.) are the names of the respective absolute elements.

In general the option field has meaning only for the particular processor, though there are some options that have the same meaning for all processors. The format of the options parameter is described in 3.2.2.

The param-1, param-2,...param-n parameters contain information supplied to the processor. With the exception of the DATA processor (see Volume 3-Section 6), which works only with files and therefore assumes filenames, the parameter fields are assumed to be in element name form although they need not represent element names. The meaning of the parameter fields is determined by the processor although the following rules are followed by the processors supplied by Sperry Univac:

1. If a field intended to contain the name of a program file is not specified, TPF$ is assumed.

2. If a field is to contain an element name and the element name is specified (but not the filename) and there is no leading period, TPF$ is assumed. If there is a leading period, then the filename is taken from the previous field, provided that the field exists and was intended to name an element or a program file; otherwise TPF$ is assumed.

The source language processors (ASM, COB, FOR, ALG, etc.) have a common interpretation of several options as well as the first three parameters. The typical standard language processor control statement takes the form:

```
@COB          SI,RO,SO
```

where SI, RO, and SO represent eltname-1, eltname-2, and eltname-3.

The meanings of these parameters are:

SI (Source Input)	If a new symbolic element is being introduced from the runstream, this parameter specifies the file into which the new element is placed and the name which it is given. If an update is being performed, then this parameter specifies the element and the cycle of the element being updated.
RO (Relocatable Output)	This parameter specifies the name and the program file into which the element produced by the processor is placed. There is no restriction on the type of element being produced. For example, most of the processors produce relocatable binary elements; the Collector produces either absolute or relocatable binary elements.
SO (Source Output)	This parameter specifies the name and the file for the updated symbolic element produced.

For each file specified in the SI, RO, and SO fields the processor dynamically performs an @ASG,AX. The file(s) are assigned to the run with exclusive use. This is necessary because the language processor is modifying the table of contents of the file(s) and/or writing in the text portion of the file(s). If another run has any of the files already assigned with exclusive use, a facility reject occurs and the compilation terminates. This problem would occur when cataloged files are referenced, and the termination of a compilation may be avoided by placing an @ASG,AX control statement for the cataloged file(s) specified in the runstream prior to the language processor control statement.

The source input routine options assist in specifying the type of processing to be performed. (See Volume 4-Table 2-13.)

If no element name is specified for RO and SO or the parameter is left blank the following rules apply:

1. If there is no file information and the parameter does not have a preceding period or if the parameter is void, then the file specified in the SI parameter is assumed.

2. The element name from the SI parameter is assumed.

3. If there is no version specified, then the version from the SI parameter is used.

Tables 3-8 and 3-9 describe the valid possibilities. The I and U options along with the SI parameters determine the interpretation of the processor control statement. An error message is printed if there is any deviation from these rules. Table 3-8 is valid for example, for the ASM, COB, FOR, ALG, and CFOR language processors (require SI, RO, and SO); Table 3-9 is valid for example, for the PDP and ELT processors (require only SI and SO).

Table 3-8. Processors that Use the SI, SO, and RO Parameters

I or U Option	SI Notes	RO Notes	SO Notes	Element Produced
Neither or I Only	Not specified	This parameter may or may not be specified. If it is not specified, NAME$ is assumed. It is invalid to specify a cycle.	Illegal to use this parameter.	New relocatable element.
Neither	Parameter is completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	If void, no source output is produced. If this parameter is not completely specified then information from the SI parameter is used. It is valid to specify a cycle.	New relocatable element. Also symbolic element if SO was specified.
I Only	Parameter is completely specified but without a cycle.	If not completely specified, the information from SI parameter is used. It is invalid to specify a cycle.	Illegal to use this parameter.	Relocatable and symbolic elements.
U Only	Parameter must be completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	New relocatable element and updated symbolic element.

Table 3-9. Processors that Require the SI and SO Parameters

I or U Option	SI Notes	SO Notes	Element Type Produced
Neither	Not specified.	Illegal to use this parameter.	The runstream input is processed but no element is produced.
Neither	Parameter is completely specified. If SO is void, the L option is assumed.	If this parameter is void, no source output is produced. If it is not completely specified, then the information from the SI parameter is used. Invalid to specify a cycle.	Update (no cycling)
I Only	Not specified.	Illegal to use this parameter.	No element is produced.
I Only	Parameter is completely specified but without the cycle.	Illegal to use this parameter.	New element.
U Only	This parameter must be completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	Update (with cycling)

3.10. DYNAMIC RUNSTREAM MODIFICATION

3.10.1. Dynamic Runstream Expansion (@ADD)

Purpose:

Provides a means of inserting images into a runstream from any file currently assigned to the user of any cataloged file, provided that it is a sector-formatted file in SDF, or from any symbolic element of a program file created by such means as the:

- DATA processor (see Volume 3-Section 6)
- ED processor (see SPERRY UNIVAC 1100 Series Text Editor (ED Processor) Programmer Reference, UP-8723 (current version))
- ELT processor (see Volume 3-Section 5)
- CTS processor (see Sperry Univac Series 1100 Conversational Time Sharing (CTS) System Programmer Reference, UP-7940 (current version))
- user program

All parameters on the @ADD control statement are optional except name.

Format:

@label:ADD,options name

Parameters:

options — The options are:

D – Allows the insertion of files or elements when operating under the DATA or ELT,D processors.

E – Return control at the EOF address of the READ$ request as if an @EOF control statement had been encountered when the end of the added file or element is reached (see 5.2.1).

NOTE:

If the DATA or ELT,D processors are executed at the time of the @ADD with the E-option, the EOF return will be ignored.

L – Use in demand mode only. Will list all control statements encountered in an added file or element at the demand terminal until the runstream returns to the demand terminal input. @ADD's nested within the @ADD,L runstream also have control statements listed.

P – The @ADD control statement is to be printed in the program listing.

name — Names the file or element to be added (see 2.6). A filename must be followed by a period, otherwise it is interpreted as an element name. If filename is not specified, TPF$ is assumed.

Description:

When the @ADD control statement is encountered, the first image of the file or element being added replaces the @ADD control statement image. All subsequent runstream images are taken from the file or element being added until either end-of-file or end-of-element is encountered. Subsequent images are obtained from the file in which the @ADD control statement was encountered.

This control statement is a valuable tool for remote users (batch or demand) because control statements or data need be entered only once but may be used in many subsequent runs. The prestored, partial runstreams can be corrected prior to their addition by placing correction lines after an @ELT,D, @ED, or an @DATA control statement.

Care should be taken to ensure that the added element or file does not contain statements which change the structure of the referenced file; e.g., @PACK, @DATA or @ERS.

Examples:

```
1.  @ADD        .TOTALS
2.  @ADD        PROFITS.
3.  @ADD,P      PROFITS.
```

1. The images in file TPF$, element TOTALS replace the @ADD control statement.

2. The images in file PROFITS replace the @ADD control statement.

3. The images in file PROFITS replace the @ADD control statement. The @ADD control statement is to appear in the program listing (P option).

3.10.2. Conditional Statements

Conditional statements are those Executive control statements which are used for the dynamic adjustment of the control stream while it is being executed. The conditional control statements provided are:

@SETC (see 3.10.4.1)

@TEST (see 3.10.4.2)

@JUMP (see 3.10.4.3)

Through the use of these control statements, the user can set values in a condition word (3.10.4) that is maintained for each run, test that value, and depending upon the results of that test, skip a portion of the control stream, or direct an individual task of the run to vary its execution. The condition word can also be accessed or altered by either the Executive or by any of the user programs within the run. The outstanding feature of this conditional network is that it allows a given runstream to produce many different results with only minor modifications to the stream.

3.10.3. Statement Labeling

Every control statement, including those registered by a CLIST$ Executive Request (see 5.5) may have a label specified. The label provides the means by which portions of the runstream can be skipped with control passed to the statement having a particular label. Note that labels contained on the @COL, @END, @ENDF, and @FILE control statements cannot be used for passing control since these control statements are not entered in the runstream. Control statements have only one label; however, additional labels can be attached to a statement by means of the use of label statements.

A label statement is any control statement containing a label but no command or operand fields; i.e., just the recognition character (@), the label, the colon (:), and optionally a comment. The comment, if present, must be preceded by the 'space period space' sequence to denote the absence of the command and operand fields.

As an example, this @XQT control statement can be referenced by any of the labels TAG, MARK, or LAST, which are attached to it by label statements. As each label statement is encountered, no operation is performed and runstream processing continues until a control statement with a command is found.

```
@TAG:
@MARK:
@LAST:XQT    PROGX
```

If the same label appears more than once within a run, the first forward occurrence from the point of reference is taken as the proper label.

3.10.4. Condition Word

The condition word format is:

T1	T2	T3
system-condition-bits	0-or- value-set-by-@SETC	0-or- value-set-by-ER-SETC$

The condition word is divided functionally into thirds, as follows:

- T1 is set by the Executive to indicate various error conditions and states with the following bit settings:

  Bit 31 - Run has been rescheduled following a recovery boot. (R option on @RUN.)

  30 - Inhibit run termination when a program error terminates (set by @SETC,I and cleared by @SETC,A, see 3.10.4.1).

  26 - Most recent activity termination was an abort termination (not error termination).

  25 - Most recent activity termination was an error termination.

  24 - One or more previous activity terminations of the current task (previous task if between executions) was an error termination (see 4.3.2).

  For example, a value of 4 in S2, between tasks, means that the last activity of the previous task did an ABORT$ request and that all other activities of that task (if any) terminated normally. Note that bits 26 and 25 cannot both be set.

- T2 is set by the @SETC control statement (see 3.10.4.1). It may also be set by means of the set parameter on the @START control statement (see 3.4.3).

- T3 is set by means of the SETC$ request (see 4.4.1).

While the entire condition word may be examined, either in the runstream (@TEST - see 3.10.4.2) or by an executing task (COND$ - see 4.4.2), alteration is limited to individual thirds, where T1 can be modified only by the Executive; T2 only by a control statement; and T3 only by an Executive Request.

The condition word is set to all zeros at the start of a run, unless:

1. the run was started by an @START control statement with a set parameter, in which case T2 initially contains the value of that parameter, or

2. the run has been rescheduled across a recovery boot through use of the R option on the @RUN control statement, in which case bit 31 is set.

The runstream path may be varied using the condition word in combination with the @SETC, @TEST, and @JUMP control statements (see 3.10.4.1, 3.10.4.2, and 3.10.4.3, respectively).

3.10.4.1. Condition Word Control (@SETC)

Purpose:

Stores (set) a value in the T2 portion of the condition word.

All parameters are optional on the @SETC control statement.

Format:

```
@label:SETC,options  f/value/j
@label:SETC,options  value/j
```

Parameters:

options — The options are:

- A – Clears bit 30 of the condition word which allows a normal ERR$ termination of this run. For example, if an error termination occurs, the run is terminated after processing @PMD and conditional control statements. This option is in effect when the run is initiated (opened).
- I – Sets bit 30 of the condition word which inhibits run termination after a program error terminates. Normal processing continues after any error terminations that occur while this option is in effect.

The I option should not be specified unless the user is willing to assume SUP costs beyond time of error detection. This option permits runs with independent tasks or tasks with nonfatal errors to continue. It also permits tests in which error conditions are expected to be encountered, to continue. These options are meaningful only for batch jobs.

f — Specifies the type of operation to be used to alter T2 of the condition word. Permissible entries for f are AND, OR, XOR. This parameter is optional. If f is specified, the result of the logical operation between T2 of the condition word and the value specified on the control statement is stored in that portion of the condition word specified by the j parameter.

value — Specifies a positive octal value not exceeding four digits in length to be used in the logical operation specified by f or to be entered into the designated portion of the condition word. Value is right-justified, zero filled prior to a partial word store into the designated portion. If the magnitude of value exceeds the capacity of the designated portion, truncation occurs.

j — Designates the portion of the condition word which is to be changed. Permissible entries are: T2, S3, or S4. If omitted, T2 is assumed.

Examples:

```
1.  @SETC       6
2.  @SETC       10/S3
3.  @SETC,I     4/S4
```

1. Loads 6_8 into T2 of condition word. (T2 is assumed since j parameter is omitted.)

2. Loads 10_8 into S3 of condition word. The value is treated as octal even when the leading zero is omitted.

3. Loads 4_8 into S4 of condition word. Set bit 30 of condition word.

3.10.4.2. Condition Word Testing (@TEST)

Purpose:

Tests the value of the condition word to select the particular control statements to be executed or skipped.

All parameters on the @TEST control statement are optional except the first occurrence of f and value.

Format:

```
@label:TEST    f/value/j,f/value/j...
```

Parameters:

f — Specifies the type of comparison to be made. If the test is met, the next control statement is skipped. If test is not met, the next control statement is executed. Permissible entries are:

- TE – Test for equal
- TNE – Test for not equal
- TG – Test for greater than
- TLE – Test for less than or equal
- TEP – Test for even parity
- TOP – Test for odd parity

value — Specifies a positive, octal value not exceeding 12 digits to be compared with that portion of the condition word specified by the j parameter.

j — Specifies the portion of condition word to be tested. Permissible entries are: W, H1, H2, T1 through T3, and S1 through S6. If omitted, T2 is assumed. Tests with j parameter of T1, T2, or T3 result in sign extension of that portion of the condition word for the specified test.

Description:

When more than one test is to be made, the control statement is scanned until a test is met or all parameters are exhausted. When a test is met, the control statement immediately following the @TEST control statement is skipped.

Examples:

```
1.  @TEST       TE/6/S3
2.  @TEST       TG/6,TE/4
3.  @TEST       TG/10/H2
4.  @TEST       TLE/4/T2
5.  @TEST       TE/4000/T2
6.  @TEST       TOP/20/T3
```

1. S3 of the condition word is tested to see if it is equal to a value of 6_8; if equal, the next control statement in the stream is skipped. If unequal, the next control statement is executed.

2. T2 of the condition word is tested for two conditions: greater than 6_8 or equal to 4_8. If either condition is met, the next control statement in the stream is skipped. If neither condition is met, the next control statement in the stream is executed.

3. H2 of the condition word is tested for a greater than 10_8 condition. If the condition is met, the next control statement in the stream is skipped; otherwise, the next control statement is executed. The value is treated as an octal value even if the leading zero is omitted.

4. T2 portion of the condition word is tested to see if it is less than or equal to 4_8. If so, the next control statement in the stream is skipped; otherwise, it is executed.

5. Due to sign extension of T2, this test always results in the next control statement being skipped. The correct test would be:

   ```
   @TEST   TE/777777774000/T2
   ```

6. T3 of the condition word is tested to see if the bit represented by 020 is set. If it is, the next control statement in the stream is skipped; otherwise, it is executed.

3.10.4.3. Branching from within Runstream (@JUMP)

Purpose:

Advances control to the specified labeled statement within the control stream.

All parameters in the @JUMP control statement are *required* except the statement label.

Format:

```
@label:JUMP   label
```

Parameters:

label — Specifies the label or name attached to a subsequent control statement to which control is passed. This parameter may instead be a decimal value indicating the number of labeled control statements to be advanced. Those control statements which cannot have labels

must not be included in the count. A numeric value of 0 as a parameter is not permitted.

Description:

The @JUMP control statement must refer in the forward direction to a statement not yet processed.

Examples:

```
1.  @JUMP       TAG
2.  @JUMP       4
```

1. Control is advanced to the control statement containing the label TAG.

2. Control is advanced to the fourth control statement, capable of having a label field, following the execution of the @JUMP control statement.

3.10.4.4. Conditional Runstream Example

```
1.  @RUN        RUNID,ACCT,PROJ
2.  @SETC       6
3.  @TEST       TE/6
4.  @XQT        A
5.  @TEST       TE/6,TE/3
6.  @JUMP       2
7.  @JUMP       X
8.  @TEST       TE/10,TE/4
9.  @JUMP       3
10. @XQT        B
11. @JUMP       Y
12. @TEST       TE/11
13. @JUMP       Z
14. @XQT        C
15. @X:XQT      D
16. @Y:XQT      E
17. @Z:XQT      F
18. @FIN
```

As shown in this example, lines 1, 2, 3, 5, 7, 15, 16, 17, and 18 are processed in this sample runstream, and programs D, E, and F are executed in that order.

Line 2 might be changed to set other values to produce different runstream processing, as follows:

Line 2	Lines Processed	Programs Executed
@SETC 3	1,2,3,4,5,7,15,16,17,18	A,D,E,F
@SETC 4(or 10)	1,2,3,4,5,6,8,10,11,16,17,18	A,B,E,F
@SETC 11	1,2,3,4,5,6,8,9,12,14,15,16,17,18	A,C,D,E,F
@SETC 1	1,2,3,4,5,6,8,9,12,13,17,18	A,F

SPERRY UNIVAC Series 1100
Executive System, Volume 2
EXEC Level 36R2
Programmer Reference
UP-4144.23-A

UPDATE SUMMARY SHEET
January 1980

UPDATE PACKAGE A

File pages as specified below

SECTION	DESTROY OLD PAGES	FILE NEW PAGES
Cover/Disclaimer	†	†
Page Status Summary	1	1
Preface	1 and 2	1 and 2
Appendix D	7 thru 12	7 thru 12
User Comment Sheet	Green	Green

† *Destroy old cover and file new cover.*

NOTES:

1. *Refer to the Page Status Summary for the latest level of each page.*

2. *When the manual has been updated in accordance with the preceding instructions, this Update Summary Sheet should be filed after the Contents section to maintain a record of updating.*

4. Executive Service Requests

4.1. INTRODUCTION

This section discusses the fundamental interfaces between an executing program and the Executive. These interfaces are the Executive Request (ER) mechanism, and the contingency mechanism. The material in this section is primarily of interest to the assembler language programmer.

The Executive Request instruction has the general form:

ER function-id

The function-id is coded in the instruction u-field as a symbolic, system-defined name. During program collection, this name is converted into an absolute ER index. The numeric index associated with each name is specified in the system-supplied element ERU$.

The function-id identifies to the Executive a particular service to be performed for the requesting activity. This service may be as simple as a clock reading, or as complex as a file handling operation. See 4.2 for a summary of all ERs provided by the Executive.

4.1.1. Coding Restrictions

Only the u-field may be used when executing an ER instruction. No indexing, index modification, or indirect addressing is permitted. Also, an ER instruction must not be executed remotely with an Execute (EX) instruction. These restrictions do not apply to the 1100/80.

4.1.2. Calling Sequence Conventions

The calling sequences for ERs are, in general, reentrant. This means that parameters are passed directly in control registers or indirectly in a packet whose address is passed in a control register. The use of Test and Set (TS) instructions across an ER to achieve reentrancy is poor practice, and in the case of real-time activities may cause a system stall which can only be relieved by operator intervention.

ER parameters that specify numeric values such as packet lengths are octal unless otherwise stated.

Normally, when a control register is required in an ER calling sequence to hold parameters or results, register A0 is used. Additional registers, if needed, are usually allocated in the sequence A1,A2,...

Control register contents (including parameters) are not altered by the execution of an ER, unless specific resultant values are defined in a control register.

The coding sequences shown for particular ERs are optimal in most instances, but any coding sequence that achieves the same register loading is permitted. Note that many ERs require just a packet address in register A0, which means only H2 of register A0 is significant; however, if the coding sequence given clears H1 of register A0 to zero, then it must be zero. This principle applies to all calling sequences.

When a parameter must be the address of another parameter, the second parameter must not reside in a control register.

4.1.3. ER Synchrony

For most ERs, processing is completed before returning control to the requesting activity. Such requests are divided into two types: synchronous and immediate. Immediate ERs are of a short, simple nature and do not normally cause switching, whereas synchronous ERs are of sufficient duration or complexity to require suspension of the requesting activity while various Executive components perform the requested service. With the exception of timing considerations pertinent to real-time applications, the differences between synchronous and immediate ERs does not influence programming logic.

A few ERs return control to the requesting activity prior to completing the required service. These ERs are asynchronous ERs. In general, control is returned immediately after processing has been initiated. The activity may then do other processing in parallel. At some future point, the program (usually the same activity) must synchronize itself with the completion of the requested service. This is most commonly done by checking a status value in a packet associated with the request. Asynchronous ERs are generally those which perform I/O.

Table 4-1 specifies the type of each ER. See Appendix B for a summary of ERs.

4.1.4. Error Handling

Programming errors in an executing program are detected in two basic ways. The first method is hardware detection of errors such as divide fault, illegal operation, guard mode violation, and so on. The second method is software detection, within the Executive, of errors in ER usage.

Such errors cover a wide range, from simple mistakes, like forgetting to assign a file, to such subtle errors as allowing all activities to deactivate waiting for each other to do something.

In a few error cases, it is not feasible for the Executive to do anything but abort the run. However, the vast majority of errors are at least potentially recoverable.

In the case of recoverable errors, the offending activity is placed in error mode, at which point a contingency occurs and the activity is either error terminated or, if a contingency routine has been registered to handle error mode contingencies, the activity is given control at that routine. See 4.9 for details of contingency handling and error termination.

In most cases, errors are detected immediately. However, certain kinds of errors for asynchronous ERs may be detected after control has been returned from the ER, in which case the contingency may occur at (and capture) an instruction address far removed from the offending instructions.

Error mode errors are not to be confused with errors not normally attributable to programming errors such as a parity error on an I/O operation. These do not cause error mode contingencies.

Documentation of the individual ERs in this manual gives all restrictions and warnings pertinent to their use, but generally does not include all possible associated error cases and error codes. In many cases, errors are common to many unrelated ERs. For example, all packet addresses are checked against program storage limits. See Appendix C for a complete list of status codes and diagnostic messages.

4.2. SUMMARY OF EXECUTIVE REQUESTS

Table 4-1 lists the name, octal function code, description, ER type, and a cross-reference for each ER. ERs that are fundamental to activity and program control, and the miscellaneous ERs are covered in this section. The remaining ERs are described in the sections covering the specific area with which the ER is associated and transaction processing manuals, etc.

Table 4-1. Available ERs

ER Name	Octal Function Code	Description	*Type	Cross Reference
ABORT$	12	Abort run	S	4.3.2.3
ABSAD$	30	Access to downed main storage	S	4.7.4
ACSF$	140	Dynamic Control statement request ASCII	S	4.10.1.2
ACT$	147	Activity activation	S	4.3.3.4
ADACT$	154	CADD$ and ACT$ (ESI only)	I	9.5
ADED$	161	Dedicate this activity to a specific processor	I	4.3.1.3
APCHCA$	77	ASCII punch control alternate	S	5.4.8
APCHCN$	75	ASCII punch control	S	5.4.6
APNCHA$	73	ASCII punch alternate	I/S	5.3.8
APRINT$	70	ASCII print	I/S	5.3.2
APRNTA$	71	ASCII print alternate	I/S	5.3.4
APRTCA$	76	ASCII print control alternate	S	5.4.4
APRTCN$	74	ASCII print control	S	5.4.2
APUNCH$	72	ASCII punch	I/S	5.3.6
AREAD$	166	ASCII read	I/S	5.2.2
AREADA$	167	ASCII read alternate	I/S	5.2.4
ATREAD$	170	ASCII print and read	I/S	5.2.6
AWAIT$	134	Wait for other activities to terminate	S	4.3.3.1
BANK$	160	Acquire Bank Descriptor Index and reload common bank	S	4.8.4
BDSPT$	115	DN mass storage granule	S	4.7.6
CADD$	57	Add communications buffer	I	9.4.2.3
CEND$	100	Terminate contingency status	I	4.9.4.2
CGET$	56	Get communications buffer	S	9.4.2.2
CJOIN$	151	Expand communications buffer pool	S	9.4.2.4
CLIST$	153	User access to control statements	S	5.5
CMD$	51	Dial communications line	A	9.4.1.2
CMH$	52	Hang-up communications line	A	9.4.1.9
CMI$	47	Initiate communications input	A	9.4.1.3
CMO$	50	Initiate communications output	A	9.4.1.4
CMS$	45	Line terminal initiation	S	9.4.1.1
CMSA$	53	Initiate communications input and output	A	9.4.1.5
CMT$	46	Terminate communications line	S	9.4.1.10
COM$	10	Console output and solicited input	S	4.6.1
COND$	66	Retrieve condition word	I	4.4.2

Table 4-1. Available ERs (continued)

ER Name	Octal Function Code	Description	*Type	Cross Reference
CPOOL$	55	Create communications buffer pool	S	9.4.2.1
CQUE$	117	Queue a contingency from CDB for user program	S	4.9.6.3
CREG$	212	Register contingency routine	I	4.9.3.2
CREL$	152	Release communications buffer pool	S	9.4.2.5
CRTN$	35	Remove activity from contingency and return	S	4.9.4.3
CSF$	17	Control statement function	S	4.10.1.1
C$TS	123	Clear test and set and notify EXEC	I	4.3.4.4
C$TSA	124	Clear test and set and activate	I	4.3.4.6
C$TSQ	122	Clear test and set and queue	S	4.3.4.5
DACT$	150	Activity deactivation	S	4.3.3.3
DATE$	22	Retrieve time and date	I	4.5.1
EABT$	26	Error terminate run	-	4.3.2.4
EDJS$	4	Return edited jump stack	S	4.8.6
ERR$	40	Error terminate activity	-	4.3.2.2
ERRPR$	202	Error message printout	S	4.10.5
EXIT$	11	Normal activity termination	-	4.3.2.1
FACIL$	114	Retrieve file assignment information	S	7.2.7
FACIT$	143	Retrieve file assignment information	S	7.2.7
FITEM$	32	Retrieve file assignment information	S	7.2.6
FORK$	13	Create new activity	S	4.3.1.1
IALL$	101	Register contingency routine	I	4.9.3
IDENT$	34	Retrieve activity identity	I	4.3.3.6
II$	27	Wait for unsolicited console input	S	4.6.2
INFO$	116	Miscellaneous information retrieval	S	4.8.7
INT$	33	Asynchronously interrupt named activity	A	4.3.3.5
IO$	1	Initiate I/O	A	6.3.4
IOADH$	206	Initiate arbitrary device I/O and wait	S	6.9.2
IOARB$	21	Initiate arbitrary device I/O	A	6.8.2
IOAXI$	20	Exit and initiate arbitrary device I/O with interrupt activity	S	6.8.3
IOI$	2	Initiate I/O with interrupt activity	A	6.3.5
IOW$	3	Initiate I/O and wait	S	6.3.6
IOWI$	24	Initiate I/O with interrupt activity and wait	S	6.3.7
IOXI$	25	Exit and initiate I/O with interrupt activity	S	6.3.8
LABEL$	31	Manipulate tape labels	S	7.3.4
LCORE$	44	Release program storage	S	4.7.2
LEVEL$	207	User activity leveling	I	4.3.7
LOAD$	111	Load program segment	S	4.7.5
LOG$	210	Create user-formatted log entry	S	4.8.8
MCORE$	43	Acquire program storage	S	4.7.1
MCT$	41	Retrieve master configuration table	S	4.8.3
NAME$	146	Name an activity	S	4.3.3.2
NRT$	62	Terminate real-time status	I	4.3.5.2
OPT$	63	Retrieve options	I	4.8.1
PCHCA$	165	Punch control alternate	S	5.4.7
PCHCN$	164	Punch control	S	5.4.5
PCT$	64	Program control table retrieval	I	4.8.2
PFD$	106	Delete an element from a program file	S	3-11.3.1.3
PFI$	104	Insert an element into a program file	S	3-11.3.1.1
PFS$	105	Find an element in a program file	S	3-11.3.1.2

Table 4-1. Available ERs (continued)

ER Name	Octal Function Code	Description	*Type	Cross Reference
PFUWL$	107	Update next program file write location	S	3-11.3.1.4
PFWL$	110	Obtain next program file write location	S	3-11.3.1.5
PNCHA$	145	Punch alternate	I/S	5.3.7
PRINT$	16	Print	I/S	5.3.1
PRNTA$	144	Print alternate	I/S	5.3.3
PRTCA$	155	Print control alternate	S	5.4.3
PRTCN$	137	Print control	S	5.4.1
PSR$	157	Processor state register control	I	4.10.2.3
PUNCH$	130	Punch	I/S	5.3.5
READ$	15	Read	I/S	5.2.1
READA$	42	Read alternate	I/S	5.2.3
ROUTE$	133	Line terminal transfer	S	9.4.3
RSI$	112	CCR interface	S	8.7
RT$	61	Establish real-time status	I/S	4.3.5.1
SETBP$	156	Set 1110, 1100/40, 1100/80 programmable breakpoint register	S	4.10.4
SETC$	65	Set condition word	I	4.4.1
SMU$	211	Communicating with the SMU	I	4.7.7
SNAP$	126	Snapshot dump	S	4.10.3
SYMB$	200	Packet driven symbiont ER	I/S	5.6
SYSBAL$	176	Turn SIP on and off from a user program	S	4.8.5
TLBL$	142	Reading and writing tape label blocks	S	7.3.3
TDATE$	54	Retrieve time and date	I	4.5.2
TFORK$	14	Create new activity with timed wait	S	4.3.1.2
TIME$	23	Retrieve time of day	I	4.5.3
TINTL$	136	Initialize tape file to beginning of first reel	S	7.2.8
TREAD$	102	Print and read	I/S	5.2.5
TRMRG$	120	Register a common bank for control at Activity/Program Termination	S	4.9.6.5
TSQCL$	113	Deregister test and set queuing for programs	I	4.3.4.3
TSQRG$	121	Register test and test queuing for program	I	4.3.4.2
TSWAP$	135	Swap reels of tape file	S	7.2.9
TWAIT$	60	Timed wait	S	4.3.6
UNLCK$	67	Terminate interrupt activity status	S	6.3.9
WAIT$	6	Wait for completion of I/O request	S	6.3.1
WALL$	37	Wait for any I/O completion return to user	S	6.3.2
WANY$	7	Wait for any I/O completion	S	6.3.2

* *Type A = Asynchronous; S = Synchronous; I = Immediate; - = Not applicable*

4.3. ACTIVITY AND PROGRAM CONTROL

4.3.1. Activity Registration

4.3.1.1. Create a New Activity (FORK$)

Purpose:

Register and initiate an asynchronous program activity.

Format:

```
L    A0,(parameter-word)
ER   FORK$
```

Description:

Each activity of a program executes independently of all other activities. A FORK$ request creates a new activity and schedules it for execution. Parameter-word describes characteristics to be associated with the new activity and specifies the program address at which it is to be given control. The format of parameter-word is:

[RT-priority]	[activity-id]	registers	entry-address

Entry-address is the program address at which the new activity is to begin execution.

The registers field specifies the set of control registers which must be saved for the activity and which initially have the same contents as the corresponding registers of the initial activity. A value of zero indicates that only the minor set of registers (X8-X11, A0-A5, R1-R3) are required. A nonzero value indicates that all X-, A-, and R-registers (except X0, A15+3, A15+4, and R0) are required. Once selected, the control register set remains with the activity until it is terminated. Note that if an activity with only the minor set of registers creates an activity with the complete register set, the initial register contents are only defined for the minor set. The space and time required, within the Executive, to maintain a minor register set activity is significantly less than for an activity with the entire register set.

The activity-id field is used to associate a numeric identity with the new activity. If used, the activity-id must be unique within the program and have a value from 1 through 35 and must not currently be in use. A zero specifies that the activity is not to have an activity-id (note that the initial activity of a program has no activity-id). See the discussion of AWAIT$ (4.3.3.1) for the use of activity-ids.

The RT-priority field allows a real-time priority to be assigned to the new activity. If used, the value must be in the range 2 through 35 and within the range allowed to the account number. Note that at least one other activity of the program must previously have elevated itself to real-time by an RT$ request (4.3.5.1) before this method can be used.

See 4.3.5 and Section 10 for additional information on real-time activity/program control and real-time processing.

4.3.1.2. Create a New Activity with Timed WAIT (TFORK$)

Purpose:

Creates a timed activity. A TFORK$ request is similar to a FORK$ request (see 4.3.1.1) except that the new activity does not begin execution for a specified amount of time.

Format:

```
L    A0,(parameter-word)
L    A1,(wait-time-in-milliseconds)
ER   TFORK$
```

Description:

The format and meaning of the parameter word is identical to the parameter-word used in a FORK$ request (see 4.3.1.1). The wait time may be any value from 2 to 30,000 (2 milliseconds to 30 seconds). The maximum value is 30,000 milliseconds and, if exceeded, 30,000 milliseconds is used. If the new activity is real-time, the wait time may exceed the 30,000 millisecond limit. Note that the wait time is simply a minimum elapsed time "by the clock", and is not influenced by the amount of processing (if any) devoted to other activities of the program (e.g., the program might be swapped). For this reason, the TFORK$ request is primarily intended for real-time applications.

4.3.1.3. Activity/Processor Dedication (ADED$)

Purpose:

To register an activity's requirement to be processed by a single, designated processor.

Format:

```
L,U  A0,processor mask
ER   ADED$
```

Description:

The processor mask is a bit mask indicating on which processor the activity is to be run. Bit 2^n indicates processor n. At most one processor may be requested. A mask setting of zero removes any activity dedication.

Subsequent processing of the activity is only on the requested processor, until such time as the activity undedicates itself or rededicates to another processor, or until such time as the operator downs the requested processor. A mask setting of zero returns the activity to normal processing mode, wherein it may be handled by any available processor. While dedicated, an activity continues to receive control as its normal switching priority. (See 12.5.6 for a discussion of switching priority.) If the requested processor is down or nonexistent or is downed subsequently by the operator, the activity is terminated in error with an error type 4 and error code 077. A mask of currently available processors appears in the second word of the contingency packet if the error mode contingency is registered. A mask of currently available processors may be obtained from the Master Configuration Table via an ER MCT$ (see 4.8.3).

A dedicated activity doing an ER FORK$ (4.3.1.1) creates an activity dedicated to the same processor.

The ER ADED$ should not be used unless necessary (hardware integrity testing, etc.) because it generally reduces the frequency with which an activity receives control. In an environment where

a large number of activities is dedicated to a single processor, switching overhead is increased slightly, and resources may be utilized inefficiently by creating a situation where one or more processors are idle while many activities are waiting to receive control on a single processor.

4.3.2. Activity Termination

The following ERs provide various forms of activity termination. When an activity terminates, it ceases to exist for the program and the system. The activity-id and name are released for reuse by any other activities. When the last activity of a program terminates, the program is terminated.

4.3.2.1. Activity Normal Termination (EXIT$)

Purpose:

Terminate the current activity.

Format:

```
ER   EXIT$
```

Description:

The current activity is terminated, and the program is also terminated if this is the last activity.

4.3.2.2. Activity Error Termination (ERR$)

Purpose:

Place the requesting activity in error mode (normally causes error termination).

Format:

```
ER   ERR$
```

Description:

See 4.1.4 and 4.9 for a complete discussion of error termination and error mode.

4.3.2.3. Abort Run (ABORT$)

Purpose:

Unconditionally terminate the program and the run.

Format:

```
ER   ABORT$
```

Description:

All activities of the program, including the requesting activity, are unconditionally terminated. No register dumps are provided for the activities. In addition, the run is terminated and PMD requests are not honored. For demand runs, only the program is terminated and not the run.

If an abort contingency routine has been registered, a single new activity is created after all the program's activities are terminated. The new activity is given control with a complete set (or if the last activity was using only the minor set, only the minor set is available) of control registers (contents not saved) at the program's contingency routine address. If any of the terminated activities were real-time, the new activity is given the highest real-time priority allowed to the run's account number. (See 4.9 for a discussion of contingencies.)

4.3.2.4. Run Error Termination (EABT$)

Purpose:

Unconditionally terminates all activities but allows error diagnostics.

Format:

```
ER  EABT$
```

Description:

The EABT$ request is similar to the ABORT$ request (4.3.2.3) except that register dumps are provided for all terminating activities and PMD requests are honored. Abort contingency also applies to the EABT$ request.

4.3.3. Activity Synchronization

In programs which use asynchronous activities to achieve parallel processing, it is often necessary for an activity to wait for the completion of processing which is being performed by other activities. Several ERs are provided to achieve the desired program synchronization. An alternate method of activity synchronization is provided by Test and Set Queuing, 4.3.4.

Activity synchronization is accomplished by removing the requesting activity from consideration for CPU time (deactivating it) until some other activities indicate that the desired processing is complete.

The programmer must be careful that all activities do not simultaneously go into synchronization waits (see AWAIT$ – 4.3.3.1, DACT$ – 4.3.3.3, and Test and Set Queuing – 4.3.4); if this occurs, the program and run are unconditionally aborted with an "AWAIT/DEACT AMBIGUITY" error diagnostic message.

There are two I/O requests, IOXI$ and IOAXI$ (see 6.3.8 and 6.8.3, respectively), which convert existing activities into interrupt activities. In these cases, the interrupt activity retains the activity-id and name associated with the original activity. Conversely, no activity-id or name retention occurs when a new interrupt activity is created (by IOI$ – 6.3.5, IOWI$ – 6.3.7, or IOARB$ – 6.8.2).

4.3.3.1. Joining of Activities (AWAIT$)

Purpose:

Deactivate requestor until all specified activities are terminated.

Format:

```
L    A0,(activity-id-mask)
ER   AWAIT$
```

Description:

The AWAIT$ request is used when further execution of the requesting activity is not desired until all specified activities have terminated.

The requesting activity must have an activity-id number (note this rules out initial program activity) as must all activities for which it desires to wait (see 4.3.1.1). The activities to be waited upon are specified by setting the bits in the activity-id mask corresponding to the activity-id number. (An activity with an id of 4 corresponds to bit 4 in the mask). Bit 0 must not be used. The requesting activity is deactivated until all specified activities have terminated by means of EXIT$ or ERR$. However, if the activity requesting the AWAIT$ has an interactivity interrupt contingency registered (see 4.9.3), it may be reactivated via an ER INT$ (see 4.3.3.5) from another activity.

4.3.3.2. Activity Naming (NAME$)

Purpose:

Attaches a name to an activity for identification purposes and for future referencing by ACT$, DACT$, or INT$ requests (see 4.3.3.4, 4.3.3.3, and 4.3.3.5, respectively). The attached name is not the same as that used in conjunction with an AWAIT$ request (see 4.3.3.1).

Format:

```
L,U  A0,18-bit activity-name
ER   NAME$
```

Description:

The user-supplied 18 bits in H2 of register A0 are expanded to 36 bits (full word) by the Executive. The user-supplied portion of the activity name must be unique for each named activity as the Executive does not otherwise guarantee uniqueness. The full 36-bit name which is returned in A0, must be used with subsequent ACT$ requests.

4.3.3.3. Activity Deactivation (DACT$)

Purpose:

Deactivates the calling activity which must have been previously named by the NAME$ request (see 4.3.3.2).

Format:

```
ER   DACT$
```

Description:

Reactivation of this activity requires that an ACT$ request (see 4.3.3.4) be made from some other activity, or if the activity has an interactivity interrupt contingency registered, it may be reactivated via an ER INT$ (see 4.3.3.5) from another activity.

If some other activity has performed an ACT$ request specifying the requester's name before the requester performed the DACT$ request, the DACT$ requester is not deactivated but is returned control immediately. This is necessary as there is no way for the activity performing the ACT$ request to determine if the requester has performed the DACT$ request. Control is immediately returned to the next instruction following the DACT$ reference of the deactivated activity.

4.3.3.4. Activity Activation (ACT$)

Purpose:

Activates an activity which must have been previously named by the NAME$ request (see 4.3.3.2).

Format:

```
L    A0,activity-name
ER   ACT$
```

Description:

The activity-name is the 36-bit name returned as a result of a NAME$ request (4.3.3.2). The requesting activity need not be a named activity.

The specified activity is reactivated at the instruction following its last DACT$ request.

If the activity being activated is already active, it is flagged such that when it next executes a DACT$ request, it is automatically reactivated. Only the first ACT$ request for the already active activity is recorded. Additional ACT$ requests are ignored until the activity performs a DACT$ request, at which time the reactivation flag is cleared.

4.3.3.5. Inter-Activity Interrupt (INT$)

Purpose:

Asynchronously interrupts a named activity.

Format:

```
L    A1,parameter
L    A0,activity-name
ER   INT$
```

Description:

The named activity (see 4.3.3.2) specified in A0 is interrupted and given an activity interrupt contingency (see 4.9). The activity interrupt contingency must be registered for the interrupted activity, either as a program or an activity contingency. Otherwise, the request is ignored. If the activity to be interrupted cannot be found, the caller is terminated in error with an error type 04 and error code 032. The activity calling INT$ need not be named.

The interrupted activity is reactivated from an AWAIT$, DACT$ or II$ state if necessary. In this case, the error address in the contingency packet points one location before the AWAIT$, DACT$ or II$ call, so that recovery may be done, as usual, by returning to the error address + 1. In all other cases, except jumps, the reentry address points to the last instruction executed. If the last instruction executed was a jump, the reentry address points to the destination address for the jump -1.

If the interrupted activity is in a TWAIT$, the time is allowed to expire. Multiple INT$ requests for the same activity are queued and processed serially.

The contents of A1 is placed in word 1 of the contingency packet (see 4.9.4.1).

4.3.3.6. Activity Identification (IDENT$)

Purpose:

Allow an activity to retrieve its identity.

Format:

```
ER   IDENT$
```

Description:

Returns the requesting activity's 36-bit name as assigned by a NAME$ request (see 4.3.3.2); also, the activity-id assigned to the activity by FORK$ (see 4.3.3.1). If the requesting activity is in real-time, its current priority is also returned. This information is returned in A0 and A1 in the following format:

A0

activity-name

A1

RT Priority	Activity-id	0

Any of the returned fields, which are not pertinent to the requesting activity, are zero filled. An activity that has not been named, has not been created by a FORK$ with an activity-id, and is not in real-time will have A0 and A1 set to zero upon return from IDENT$.

4.3.4. Test and Set Queuing

The Test and Set Queuing (TSQ) mechanism provides an alternative to the ACT$/DACT$/NAME$ Executive Requests (see 4.3.3) for synchronization of activities. For certain types of synchronization applications – primarily those involving data protection and transfer – TSQ is faster and more flexible than ACT$/DACT$, as it does not require activity naming.

Test and Set Queuing may be used in conjunction with common data banks to provide interprogram data protection and activity synchronization. See 4.11.7.3 for a discussion of TSQ use with common data banks.

When TSQ processing registration is requested for the user program via the TSQRG$ request (see 4.3.4.2), or for common banks as one of the configuration parameters, the Executive automatically deactivates and queues activities which encounter conflicts on specially-marked user TS cells. (See 2.5.5 for a discussion of the TS instruction and normal Test and Set processing.)

Determination of whether TSQ applies to a particular conflict situation is based upon an identifier in S2 of the TS cell (see 4.3.4.1), and upon whether TSQ is registered. Registration is determined by the location of the TS cell, and not by the location of the TS instruction. If the cell is in a common bank, TSQ must be registered for the common bank; if the cell is in a program, TSQ must be registered for the program. Common banks and program registrations are independent of each other. If TSQ is determined not to apply, standard TS action (see 2.5.5 and 10.4.3) is taken. This allows conventional TS to be used within the same program. However, if using regular TS and TSQ simultaneously, S2 of all regular TS cells must always be zero to avoid confusion with the specially-marked TSQ cells.

For real-time programs, TSQ registration is checked before real-time test and set contingency registration (see 4.9 and 10.4.3). Thus, if both are being used, TSQ prevails. If it is determined that TSQ does not apply in a certain situation, real-time TS is checked; if that does not apply, standard action is used.

There is a unique queue associated with each special TS cell. The activity which subsequently clears the TS lock must inspect the queue associated with that TS cell and notify the Executive if the queue is not empty. Three system-defined PROCs (C$TS – 4.3.4.4, C$TSQ – 4.3.4.5, and C$TSA – 4.3.4.6) are provided for this purpose. The sequence and format of the instructions generated by these PROCs are very important for the successful operation of the TSQ mechanism. Use of the associated ERs outside of the PROCs is not supported. All three PROCs assume that the Test and Set lock has been previously set by the user activity. If the TS queue is empty, no diversion of the activity clearing the cell is done.

The queue is ordered by switching priority (see 12.5.6 for a discussion of switching priority); hence, real-time activities of different levels may reference a TS cell without changing priority, and batch or demand activities may reference a TS cell also used by real-time activities.

Any attempt to suspend a program by putting all activities in a combination of AWAITS (see 4.3.3.1), DACT$ (see 4.3.3.3), or TSQ waits causes an unconditional program abort with an AWAIT/DEACT AMBIGUITY error diagnostic message. This does not apply to activities in TSQ waits within a common bank.

When a program is aborted with activities on TS queues, the Executive removes these activities from the queues before terminating them. This applies to TS cells within common data banks, as well as within program banks. Test and set queuing is deregistered for a program (but not for a common data bank) when an ABORT$ contingency (see 4.3.2.3) is given control.

Correct operation of the TSQ mechanism is not guaranteed if a different activity clears a lock other than the one which originally set it. Test and Set Queuing is not available to ESI completion activities.

4.3.4.1. TSQ Cell Format (T$CELL)

A system-defined PROC, T$CELL, is provided for the user program to generate the TS cells in the format required by TSQ.

Format:

```
tscell  T$CELL  user-value
```

Description:

The Test and Set cell generated at address 'tscell' has the followingformat (T$CELL is referenced in following sections).

ind	ident	user-value	queue indicator

where:

ind – Test and Set indicator portion. Must be zero initially.

ident – Fixed constant (057) generated by T$CELL by which the EXEC identifies the location as one for which TSQ is to apply. If this identifier is not present, standard test and set action is used (see 4.3.4).

user-value – Available for the user program.

queue indicator – Used by the EXEC to indicate activities are on the queue. Must be zero initially and should not be altered by the user program.

When an activity is queued, its reentry address is set to the Test and Set instruction, so that the Test and Set is reexecuted upon activation. When errors are detected in the test and set queue, the user activity in control is terminated with an error code of 0407.

4.3.4.2. TSQ Registration (TSQRG$)

Purpose:

To register automatic queuing of Test and Set conflicts within a user program.

Format:

```
ER  TSQRG$
```

Description:

Registers Test and Set Queuing (TSQ) for the user program. Activities which encounter Test and Set conflicts on T$CELL cells are deactivated and queued by the Executive. The ER TSQRG$ does not apply to TSQ registration for common data banks.

4.3.4.3. TSQ Deregistration (TSQCL$)

Purpose:

To deregister Test and Set Queuing for a user program.

Format:

```
ER  TSQCL$
```

Description:

Returns Test and Set conflict processing for a user program to normal mode (see 2.5.5 and 10.4.3). If TSQ mode is cleared with activities still queued, they remain queued until a CTS, CTSA, or C$TSQ (see below) is done.

The ER TSQCL$ does not apply to common banks.

4.3.4.4. Clear Test and Set and Notify Executive (C$TS)

Purpose:

To clear a TSQ lock and, if activities are queued, notify the Executive to activate the next activity.

Format:

```
C$TS  tscell
```

Description:

The parameter tscell is the address of a TS cell generated by T$CELL. C$TS clears the TS lock and inspects the queue (tscell, H2). If the queue is not empty, an Executive Request is made to remove and activate the highest priority non-C$TSQ activity (see 4.3.4.5) from the queue. No Executive Request is executed if the queue is empty.

C$TS accommodates normal one-level indexing (e.g., C$TS tscell, X_x). Indirect addressing, index incrementation, use of the Execute instruction (EX), and instruction modification are not supported.

C$TS operates regardless of TSQ registration.

C$TS generates the instruction sequence:

```
SZ,S1    tscell
TZ,H2    tscell
ER       CTS$
```

4.3.4.5. Clear Test and Set and Queue (C$TSQ)

Purpose:

To provide, in conjunction with C$TSA (see 4.3.4.6), selective activity synchronization and data protection.

Format:

C$TSQ tscell

Description:

The parameter tscell is the address of a TS cell generated by T$CELL. C$TSQ, when used with C$TSA (4.3.4.6), provides an alternative to DACT$/ACT$ for synchronization of activities which communicate via a common data area. An activity may lock (via TS on tscell) the common area (for example, a queue of items to process), and inspect the area for input from another activity. Finding nothing, the activity may simultaneously unlock the queue and deactivate itself by performing a C$TSQ operation. The C$TSQ request generates an Executive Request which deactivates the calling activity, places it on the TS queue associated with tscell (as if it had failed a TS on tscell), and marks the activity as having done a C$TSQ. The TS lock for TSCELL is then cleared, and action similar to C$TS (4.3.4.4) is performed to reactivate any activities which may have been queued for failing a TS on tscell while this activity had it set.

Marking the activity as having done a C$TSQ is important because it suspends the activity, making it ineligible for reactivation until a C$TSA is executed for tscell. This means that other activities doing C$TS and C$TSQ operations on tscell do not cause unnecessary reactivation of C$TSQ activities awaiting specific notification (via C$TSA) that they may proceed. When a C$TSQ activity is reactivated, control is returned immediately following the C$TSQ call and the TS lock is not set.

C$TSQ accommodates normal one-level indexing (e.g., C$TSQ tscell, X_x). Indirect addressing, index incrementation, use of the Execute Remote instruction (EX), and instruction modification are not supported.

C$TSQ operates regardless of TSQ registration.

C$TSQ generates the instruction sequence:

```
NOP     tscell
ER      CTSQ$
```

4.3.4.6. Clear Test and Set and Activate (C$TSA)

Purpose:

To simultaneously clear a TS lock and remove an activity from a C$TSQ wait.

Format:

C$TSA tscell

Description:

The parameter tscell is the address of a TS cell generated by T$CELL. C$TSA must be used to reactivate an activity previously queued via C$TSQ (see 4.3.4.5). After setting the TS lock for tscell and, presumably, adding input for another activity to a common data area protected by tscell, the activity in control may simultaneously unlock the data area, and reactivate the highest priority C$TSQ activity (if any) on tscell's TS queue by performing a C$TSA operation. The C$TSA request clears the TS lock for tscell, then inspects the TS queue. If the queue is not empty, an Executive Request is executed which searches the queue for the first activity marked as having done a C$TSQ. If such an activity is found, its mark is cleared, making it eligible for reactivation, and a C$TS operation (see 4.3.4.4) is performed. If this activity is the highest priority unmarked activity on tscell's queue, it is

reactivated at this time. Otherwise, it must wait while higher priority activities receive their turns to access the common data area. If no marked activities could be found on the queue, C$TSA operation is identical to C$TS. If the queue is empty, no Executive Request is performed.

At most, one C$TSQ activity is enabled for each C$TSA call, and a C$TSA done before an associated C$TSQ is lost. That is, there is no reactivation indicator set if C$TSA finds no C$TSQ activities on the queue.

C$TSA accommodates normal one-level indexing (e.g., C$TSA CELL,$X_x$). Indirect addressing, index incrementation, use of the Execute Remote instruction (EX), and instruction modification are not supported.

C$TSA operates regardless of TSQ registration.

C$TSA generates the instruction sequence:

```
SZ,S1    tscell
TZ,H2    tscell
ER       CTSA$
```

4.3.4.7. Examples of TSQ Usage

Example 1:

A real-time program executing on a unit processor contains activity A at level-2 and activity B at level-4. B is in control, does a TS, and gets interrupted in critical code. Activity A receives control, does a TS on the same cell and is deactivated and queued, thereby giving B control. B does a C$TS to clear the TS lock and activate A. Note that B need not be real-time; it may be batch or demand.

Example 2:

Conditions same as Example 1. B does an ER CSF$ with a TS lock set and now runs at lower priority. Activity A gets deactivated on a TS conflict, thereby allowing B to finish.

Example 3:

Consider a "producer-consumer" situation with two programs, A and B. Activity A1 of program A is passing information via a common data bank to activity B1 of program B. The following subroutines handle the data protection and activity synchronization. CELL is in a common bank which has TSQ registered.

Within activity B1:

Lock data area (TS CELL).

Check for input.

If input exists, remove it, clear lock (C$TS CELL), return.

If no input, clear lock and deactivate (C$TSQ CELL).

Within activity A1:

Lock data area (TS CELL).

Deposit input for B1.

Clear lock and activate (C$TSA CELL).

Return

4.3.5. Real-Time Program/Activity Control

4.3.5.1. Changing Program/Activity to Real-Time Status (RT$)

Purpose:

Raises the status of the calling activity and its associated program to real-time if the run's account number permits such action. The RT$ request also allows a real-time activity to change its switching priority level within the real-time class.

Format:

```
L,U     A0,switching-priority-level
ER      RT$

        or

LN,U    A0,switching-priority-level
ER      RT$
```

Description:

For an activity to be raised to real-time status, it must execute an ER RT$ or be created by a FORK$ or TFORK$ request (4.3.1.1, 4.3.1.2). A program is given real-time status when at least one of its activities is real time. Real-time status is provided for programs servicing communications lines. To allow for the time-critical nature of these programs, a program/activity which is real-time is afforded privileges which non-real-time programs/activities do not enjoy. Namely, they:

- have top priority for CPU switching and I/O,
- are not swapped out of main storage,
- have access to certain ERs, primarily the communications requests.

The highest allowable switching priority [2 (highest) through 35 in absolute value] for each program's activities is controlled by account number; requests for priorities lower than 35 are given 35. If a switching priority higher than that permitted is requested, the activity is placed in error mode. When the requesting activity currently has real-time status, the RT$ request is used to control that activity's switching priority. Since a real-time program may not be swapped, the first RT$ request from within a program may cause it to be suspended and repositioned in main storage so as to avoid memory fragmentation. This repositioning is not necessary if the program has been designated as real-time load via the REAL-TIME parameter on the collector TYPE directive (see 3-2.2.2.13). Usage of the REAL-TIME parameter is encouraged as system overhead is minimized and RT$ requests are serviced most expediently for real-time load programs. The program repositioning can also be avoided if the

first activity to attain real-time status does so by using the negative of the switching level. This allows programs to attain real-time status for short durations without the overhead of positioning. However, if a positioning RT$ request is made for a non-real-time load program, at a time when all current real-time activities of the program are of the no-positioning type, the requesting activity is placed in error mode.

4.3.5.2. Removal of Program/Activity Real-Time Status (NRT$)

Purpose:

Removes an activity/program from real-time status.

Format:

```
ER  NRT$
```

Description:

The activity is returned to the original program type (transaction, batch, or demand). A program retains real-time status until all real-time activities are reduced in status or terminated, at which time the program also returns to its initial type.

4.3.6. Timed Activity Wait (TWAIT$)

Purpose:

Places the activity in a wait state (delays execution) for a specified period of time.

Format:

```
L   A1,(wait-time-in-milliseconds)
ER  TWAIT$
```

Description:

The activity must load register A1 with the desired wait time prior to making the TWAIT$ request. This value has the same meaning as the wait time for the TFORK$ request (see 4.3.1.2).

Any activity on the TWAIT$ queue of a processor, when the processor is downed by the system operation, is activated immediately.

4.3.7. User Activity Leveling (LEVEL$)

Purpose:

To allow a user activity to change its activity level to the lowest configured user activity level. To provide the ability for a site to configure a background activity level for use by online maintenance activities.

Format:

```
ER  LEVEL$
```

Description:

The switching priority level before ER LEVEL$ is returned in A0. A user activity doing ER LEVEL$ is placed at the last configured user activity level. A realtime activity using ER LEVEL$ will be given a contingency of error type 4, error code 6, and contingency type 012.

4.4. CONDITION WORD CONTROL

The program condition word which contains program status information supplied by the Executive and information inserted by user-supplied control statements can also be modified dynamically from within an executing program. A complete description of the format and contents of the condition word can be found in 3.10.4.

4.4.1. Setting the Condition Word (SETC$)

Purpose:

Dynamically sets T3 of the program condition word.

Format:

```
L,U  A0,value
ER   SETC$
```

Description:

This ER transfers bits 11-0 of register A0 to T3 of the condition word. At the beginning of a run, the contents of T3 of the condition word are 0. This ER performs a function similar to the @SETC control statement which sets T2 of the condition word (see 3.10.4).

4.4.2. Condition Word Retrieval (COND$)

Purpose:

Transfers the program condition word to register A0.

Format:

```
ER  COND$
```

Description:

The program condition word is placed in register A0. This does not modify the condition word itself. The condition word format and contents returned by the COND$ are described in 3.10.4.

4.5. RETRIEVAL OF THE TIME AND DATE

4.5.1. Time and Date in Fieldata (DATE$)

Purpose:

Places the Fieldata date and time into registers A0 and A1, respectively.

Format:

```
ER  DATE$
```

Description:

Register A0 contains:

month (01-12)	day-of-month (01-31)	last-two-digits-of-the-year

Register A1 contains:

hour (00 through 23)	minutes (00-59)	seconds (00-59)

4.5.2. Time and Date in Binary (TDATE$)

Purpose:

Places the binary date and time into register A0.

Format:

```
ER  TDATE$
```

Description:

Register A0 contains:

month	day	year (MODULO 1964)	time-in-seconds-past-midnight

S3 is the year, modulo 1964. The current year can be computed by adding 1964 to the value returned in S3 of A0.

4.5.3. Time in Milliseconds (TIME$)

Purpose:

Places the current time past midnight in milliseconds into control register A0 in binary.

Format:

```
ER  TIME$
```

Description:

Register A0 contains:

time-in-milliseconds-past-midnight	
35	0

4.6. CONSOLE COMMUNICATIONS

4.6.1. Console Output and Solicited Input (COM$)

Purpose:

To request use of the onsite operator's console to display output messages and solicit operator input.

Format:

```
L,U  A0,pktaddr
ER   COM$
```

This linkage can also be generated by the procedure call:

```
C$OM pktaddr
```

Description:

Pktaddr is the address of a packet whose format is:

00	error-code	console-class	control-bits	actual-input-char-count
01	0	output-char-count (maximum 511)		output-buffer-addr
02	expected-input-char-count (maximum 60)			input-buffer-addr

This packet can be generated by the procedure call:

C$OMPK,console-class,control-bits
output-char-count,count-buffer-addr,expected-input-char-count,input-buffer-addr

Word 0

error-code — Contains a COM$ request error code (see Appendix C) if an error is detected in the packet (the activity is also placed in error mode).

console-class — The user may direct the message to specific console devices by specifying the appropriate console class code. The codes are:

- 0 – System console messages
- 1 – I/O activity console messages
- 2 – Communications console messages
- 3 – Hardware confidence console messages
- 4–7 – Four additional message categories may also be used to direct messages to specific console devices. The class code for these categories may be defined for individual site applications of the particular Series 1100 system being used.

control bits — Bit 20 = 1 Indicates that the console message and response is in quarter-word ASCII. If bit 20 = 0, Fieldata format is assumed.

actual-input-char-count — Contains the number of input characters received. Always less than or equal to expected input character count.

Word 1

output-char-count — The number of characters in the message to be displayed. The message is restricted to 511 characters. Each character is edited and invalid characters are deleted from the message. For Fieldata requests, the character codes 00–04 are illegal. For ASCII, the character codes 00–037 may not be displayed. If this field is zero, the COM$ request is ignored. The first 54 (62 for the Executive) printable characters are displayed on the CRT. For systems with 4009 consoles, the first 72 (80 for Executive) printable characters are displayed on the console printers. For systems with 4013 consoles, the first 100 (108 for Executive) printable characters are displayed on the console printers.

output-buffer-addr — The address of the program buffer containing the output message. The characters of the message are obtained from successive sixths of a word (for Fieldata), beginning with S1 of the first word of the buffer; or successive quarter words (for ASCII), beginning with Q1 of the first word.

Word 2

expected-input-char-count — When this field contains a nonzero value, a console operator response is solicited. The activity executing the COM$ request is placed in a wait state until the input message is complete. If the input message exceeds the expected character count, the input message is discarded and the console operator is requested to retype the message. When no input message is desired, set this field to zero. The maximum number of characters permitted in the input message is 60.

input-buffer-addr — The address of the program buffer that holds the input message. Input is returned to the requestor in the same character code as that specified in the COM$ packet. Console input for both ASCII and Fieldata is stored left-justified, starting in the first word of the input buffer. If the last word of the input message does not contain six Fieldata characters, or four ASCII characters, the remainder of the word is filled with blanks (05 or 040).

4.6.2. Wait for Unsolicited Console Input (II$)

Purpose:

Provides a means to define the activity which is to accept any unsolicited console input directed to the program.

Format:

```
ER    II$
```

Description:

The activity executing the II$ request is deactivated as for a DACT$ request (see 4.3.3.3); however, the activity need not be named. If named, it may be reactivated using an ACT$ request (see 4.3.3.4). If it also has the interactivity interrupt contingency registered, it may be activated via an INT$ request (see 4.3.3.5). An II$ request is not allowed when an II$ activity has already been defined for the program.

Unsolicited console input of up to six characters in Fieldata is stored (left-justified, space filled) in the activity's A0 register, and the activity is activated.

After activation (by either ER ACT$, ER INT$ or console input), the activity is no longer defined as the unsolicited console input activity. The same activity or some other activity must execute another II$ request to redefine the unsolicited console input activity.

Unless the program is guaranteed that unsolicited input occurs to cause the activation of the II$ activity, the activity must be named and activated by an ACT$ request prior to program termination. Failure to do this aborts the program and the message "AWAIT/DACT AMBIGUITY" is placed in the program's PRINT$ file.

The console input activity is also activated by the remote terminal BREAK keyin. Since no input is actually received, register A0 is space filled.

4.7. PROGRAM STORAGE CONTROL

4.7.1. Main Storage Expansion (MCORE$)

Purpose:

Permits user program to request additional main storage for any bank.

Format:

```
L    A0,(parameter)
ER   MCORE$
```

Description:

The format of the parameter is:

BDI	highest-address-req.

This request expands the specified bank, if necessary, to a size sufficient to include the highest address requested. If the BDI (Bank Description Index) is zero, the request indicates that one of the two banks of the active PSR is to be changed. The address requested by the activity is assumed to be in the I-bank if the address is less than the first D-bank address produced in the collection. Otherwise, the address is assumed to be a D-bank address.

If the main storage requested is already assigned to the bank, control returns to the requesting activity with no other action taken. If the storage is not already assigned, the requesting activity is deactivated until storage can be made available by swapping this or some other program. The additional storage is available to all activities of the program (including ESI completion activities) when the requesting activity is returned control. All activities (regardless of program) which have a common bank based will have the expanded size available before the activity requesting expansion is returned to control.

Additional storage cannot be obtained if any activity of the program is in real-time status, unless the storage can be obtained without moving the program. A common bank may not be moved to satisfy an expansion request if any activity of any real-time program has the common bank based. Non-real-time programs are swapped, if necessary, to make the requested storage space available (see 4.7.3 for additional restrictions on the use of MCORE$).

4.7.2. Main Storage Contraction (LCORE$)

Purpose:

Releases unneeded main storage in any bank.

Format:

```
L    A0,(parameter)
ER   LCORE$
```

Description:

This request releases storage, if necessary, from the specified bank. The bank is of the smallest size necessary to contain the parameter specified address. The parameter contains the address and BDI of the bank to be changed. The format is the same as for MCORE$ (4.7.1).

If the BDI is zero, the address specified is assumed to exist in the I-bank of the active PSR if the address is less than the first D-bank address produced in the collection. Otherwise, the address is assumed to be a D-bank address.

The entire bank can be released by specifying the first address of the respective bank. Before programming a release of the control bank, however, one must ensure that necessary Collector-produced tables are not contained in the control bank.

When main storage is actually released, the segment load table is updated. If the segment is in main storage at the time of the release and any part of the segment is no longer within the program's area, the segment is marked not in main storage. If the control bank is dynamic and not currently active, the update of the segment load table is not performed. All I/O for a program must be completed before a program bank can be contracted. All I/O for all programs with the common bank based by any activity must complete before an LCORE$ for that common bank is completed.

When control is returned to the requesting activity, the storage is no longer available to any other activity (including ESI completion activities) of the program. Multiactivity programs must ensure that no other activities are currently using the space to be released, or guard mode faults will occur.

4.7.3. Restrictions on the Use of MCORE$/LCORE$

There are certain restrictions imposed on the use of MCORE$/LCORE$ to protect the program and system integrity.

The restrictions on MCORE$/LCORE$ which, when violated, take the requestor to error mode are:

1. No write-protected bank can be modified using MCORE$/LCORE$.

2. A program may not request more storage than all of user storage. The determination of total program size is made at the time of request and assumes the bank at its expanded size. The total program size is the sum of:

   a. All program static banks

   b. All program dynamic banks with a nonzero use count.

   c. All common banks to which the program is attached.

   d. The PCT bank.

3. A real-time program is not swapped to accommodate expansion.

4. The program may not release an area currently in use as communication buffer pools.

5. A common bank can not be modified with MCORE$/LCORE$ unless the requesting activity has the common bank based.

4.7.4. Main Storage Absolute Addressing (ABSAD$)

Purpose:

Allow access to main storage by absolute address.

Format:

```
L,U  A0,pktaddr
ER   ABSAD$
```

Description:

The packet pointed to by pktaddr can have two formats. ABSAD$ distinguishes between them by checking T1 of word 1.

2-word format (absolute address):

00	number of 0100 word blocks of main storage being requested	bank descriptor index of a zero length program bank
01	absolute address of start of the main storage area being requested (multiple of 0100)	

3-word format (module name) (not used on 1100/80A):

00	number of 0100 word blocks of the storage module being requested	bank descriptor index of a zero length program bank
01	logical name (Fieldata left justified space filled) of storage module containing requested area	
02	0100 word block offset from the beginning of the module to the start of requested area	

The main storage must be in a reserved state (via an operator RV keyin). Control is returned following the ER with one of the following statuses in A0:

0 - Successful completion.

1 - Improper BDI. The BDI is not defined, or the indicated bank is not zero length.

2 - Part or all of the requested main storage is not reserved, or the requested area is currently held by some other program, or the request went beyond the end of a storage module.

3 - The requested area was not accessible. Either the area contained addresses outside the main storage chain, or the module was not described in the Master Configuration Table. In the latter case, A0 increment contains the status returned to ABSAD from MCT$.

In order to set the PSR and storage limits so that the activity does not guard mode when trying to access the reserved storage, it is necessary for each referencing activity to LBJ (or LIJ or LDJ) to the bank after the main storage has been acquired. Note that having the bank initially based or doing an LBJ to the bank before acquiring the main storage does not allow the program to access the reserved main storage acquired via ABSAD$.

On 1100/80 Systems with 4x4 capability, the three-word packet format is not permitted. All requests using the three-word packet format will be rejected and a status of 02 returned in A0.

The LCORE$ request (see 4.7.2) may be used to release main storage acquired via ABSAD$. Any reserved main storage which is not released is returned to nonheld state when the program terminates. Note that the reserved status is at no time altered by being acquired via ABSAD$ or released via LCORE$.

4.7.5. Segment Loads (LOAD$)

Purpose:

Loads a segment of a program.

Format:

```
L,U  A0,seg-name      or   L    A0,(0400000,seg-name)
L,U  A1,jump
L,U  A2,rseg-addr     or   L    A2,(bank-name,rseg,addr)
ER   LOAD$
```

Parameters:

seg-name — Specifies the name of the segment to be loaded. Seg-name is the same as the contents of the name-1 parameter in the SEG directive (see Volume 3-2.2.14) or 0400000 meaning the main segment.

jump — Specifies the location where control is to be transferred after the segment is loaded; if omitted, control passes to the location following the LOAD$ request.

rseg-addr — If the segment was defined by an RSEG directive, this parameter specifies the starting address for the relocatable segment. If omitted when loading a relocatable segment, the address must be in register A2 before the call is made:

```
L,U  A2,rseg-address
```

The address may be defined as an octal value or a tag not contained in an RSEG.

bank-name — Used only for loading an RSEG within a bank-named collection. Specifies the bank into which the RSEG is to be loaded.

Description:

Whenever a segment is loaded, an initial copy of the segment is loaded. Once loaded, a segment remains marked as loaded until all or part of its main storage space is overlayed by another segment or released via ER LCORE$ (see 4.7.2).

When bit 35 of register A0 is set, the segment loader does not clear the main storage area to be occupied by the segment. This decreases the time required to load the segment, but as a result, any areas within the segment that are not initialized with data and instructions cannot be assumed to be zero.

Although the main segment of every program is always automatically loaded at the start of execution, and stays loaded throughout execution, it is possible to reinitialize a program by reloading the main segment, using the value 0400000 instead of segname. This causes all other program segments to be marked as unloaded. See Volume 3-2.2.4.5.1 for L$OAD procedure and ER LOAD$ examples.

4.7.6. Down by Track (BDSPT$)

Purpose:

Provide a means for the user to remove a track from the system's mass storage pool. Input to BDSPT$ can be a file relative granule or a list of device area descriptions.

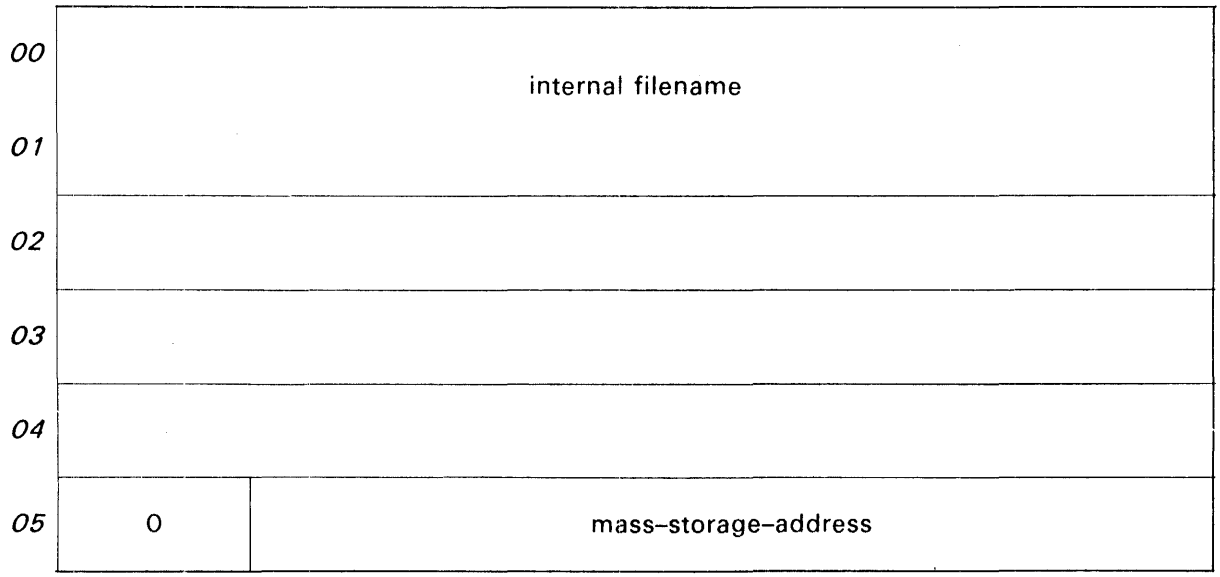
The BDSPT$ Executive Request can only be performed by privileged runs.

Format:

The format of the BDSPT$ request for file relative granule input is:

```
L     A0,(0,pktadr)
ER    BDSPT$
```

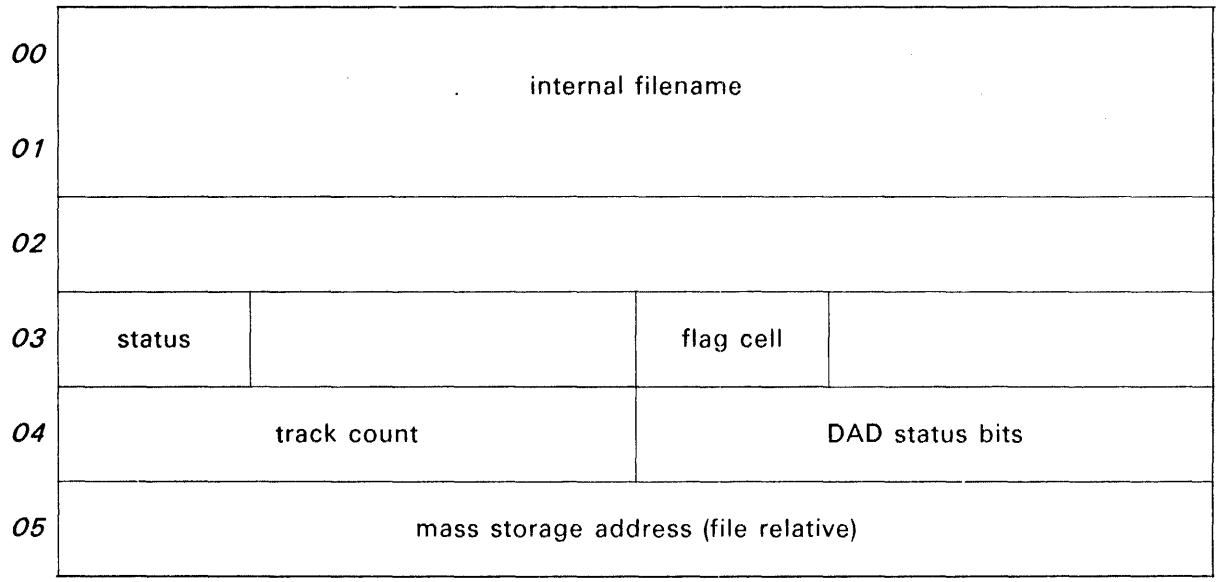
The pktaddr loaded into register A0 references a packet having the following format:



Word 5

This word contains the logical mass storage address at which the described I/O operation is to start. This address is relative to the start of the mass storage file; the handler determines the absolute address. For sector-formatted mass storage files, the address is the start of a sector, and consecutive addresses are 28 words apart.

If Bit 35 of word 5 is set, a modified packet is defined with the following format:



Word 03

Status

010 Flag cell conflict.
020 Starting file relative address bad (beyond HGA).
021 Track count bad (file relative address plus track count is beyond file's highest granule assigned, bit change is made through HGA).

Flag Cell

Indicates the type of status changes to be made. The allowable types are:

1 Set Status
2 Clear Status

Word 04

Track Count

Specifies the length, in tracks, of the area to be changed. This allows the status bits to be set or cleared in more than one DAD per request (with the exception of the BADSPOT flag).

DAD Status Bits

Master bitted status setting or clearing:

001 Badspot
010 DNR (Do Not Read)
020 DNW (Do Not Write)

Note that clearing the badspot bit is not allowed. The DNR and DNW bits are used to prevent the reading or writing, respectively, of specific file relative areas described by individual DAD entries.

Word 05

Mass Storage Address	Bit 35 set signifies new format packet. Bits 31 thru 0 identify the file relative sector address (word address for WAD file) of the area requested for status change. For DNR and DNW setting and clearing, the DAD entry which contains the requested file relative address is set or cleared. Badspot handling is not altered.

Description:

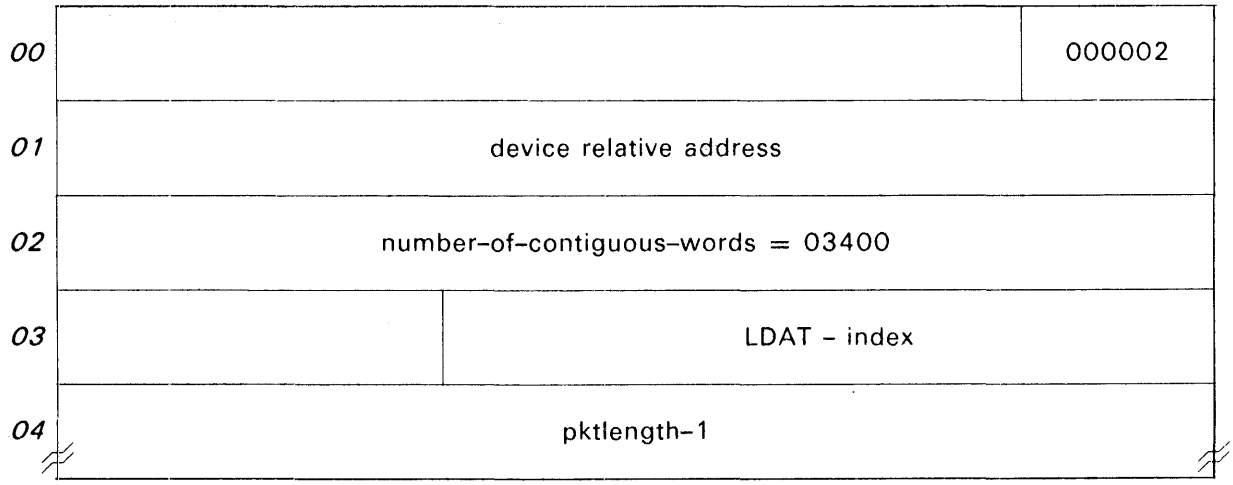
This function of BDSPT$ marks the file relative granule specified to eventually be removed from the system's mass storage pool. The actual removal of the track is delayed until the granule is released from the file, either by an I/O REL$ or by deletion of the file. It also allows the setting and clearing of DNR and DNW bits in the DAD entries.

Format:

The format of the BDSPT$ function for relative track address input is:

```
L    A0,(pktlength,pktaddr)
ER   BDSPT$
```

The pktaddr loaded into the lower half of A0 references a packet being pktlength words in length having the following format:



Word zero of the packet contains a function code which must be equal to two.

Word one of the packet is the first word of the first DAD. Device relative address is of the same type as that which appears in a master file directory granule table. Disk addresses are not accepted.

Description:

This function of BDSPT$ checks to see if the input addresses are already removed from the available mass storage pool. If the input addresses are available, the tracks are removed from the mass storage pool immediately.

When BDSPT$ returns control to the user, register A0 contains the original packet address in H2, the original contents in S3, and any possible error condition in T1.

On return, if A0 is negative, the following error conditions apply:

T1 of A0	ERROR
04001	Run doing BDSPT$ was not privileged.
04002	Pktaddr was not within program limits.
04003	User file specified not assigned.
04004	Illegal function specified.
04005	Encountered at least one illegal address. Packet words containing legal addresses are zeroed out and words containing any illegal addresses are left as they were.
04006	System problem caused ER not to be completed (e.g., system cannot assign SYS$*DOWNEDTRACKS file).
04007	An error status is returned in the packet status cell.

4.7.7. Communicating with the SMU (SMU$)

Purpose:

To allow a requestor to pass PANL directives from the 1100/80 CPU to the PANL program executing in the System Maintenance Unit (SMU) and also to allow the same requestor to terminate the passing of PANL directives by terminating IMI mode.

Format:

```
L,U  A0,PACKET
ER   SMU$
```

To cause the PANL program to accept directives from the CPU, the format of PACKET is the following:

00	04		status	
01	length			address

Word 0

04 — Indicates a request is being made to pass PANL directives from the CPU.

status — Provided upon return from ER SMU$.

The following values for status are defined:

- 0 – The directive sequence was not sent to the SMU. Either the CPU was not in maintenance mode, or PANL was not in IMI mode.
- 1 – The directive sequence was accepted by PANL.

2 - The directive sequence was rejected because PANL encountered an error fetching directives from main storage.

3 - The SMU is already in use.

Word 1

length — Length of the directive sequence in words.

address — Address of the directive sequence.

To cause the PANL program to terminate IMI mode and resume solicitation for input from the SMU work station, the format of PACKET is the following:

00	05		status	
01	length			address

Word 0

05 — Indicates that PANL is to terminate IMI mode.

status — Provided upon return from ER SMU$.

The following values for status are defined:

0 - IMI did not go through to the SMU

1 - PANL has terminated IMI mode.

Word 1

length — Length of the directive sequence in words.

address — Address of the directive sequence.

Description:

For a function value of 04, the directive sequence to be passed to the SMU must consist of PANL directives; the first one of the sequence beginning on a word boundary. Each directive in the string of directives must be terminated by a '\' character. A directive must use the Fieldata code for characters and cannot exceed 40 characters not including the '\' character. The maximum length of the directive sequence is 62 directives. The end of the directive sequence is given by two consecutive '\' characters. Some caution must be exercised in using SMU$, since a directive error terminates the execution of a directive sequence and leaves a CPU in a stopped condition.

Example:

```
'DΔCAU0'
'ΔΔΔΔΔ\'
'RΔΔ\\Δ'
```

NOTE:

The Δ character is a significant space.

The first time ER SMU$ is used, the eligibility of the user to access the SMU is checked. If use of this ER is not allowed, requests for the ER causes a contingency to be generated with error type, error code, and contingency type, respectively, of 040212. Once the ER is allowed, repeated requests for the ER are also allowed.

To cause the PANL program to resume soliciting directives from the maintenance work station, a function value of 05 is placed in the SMU$ packet. To pass PANL directives again from the CPU, the requestor must place a function value of 04 in the SMU$ packet.

Use of this ER while in contingency mode causes contingency mode to be retained.

4.8. PROGRAM/SYSTEM INFORMATION RETRIEVAL

4.8.1. Retrieving @XQT Control Statement Options (OPT$)

Purpose:

Makes available options specified on the @XQT or processor call statement (see 3.4.4).

Format:

```
ER  OPT$
```

Description:

When control is returned, the specified option letters are set in register A0 in master bit notation; that is, letter A sets bit 25; letter B sets bit 24; letter C sets bit 23;...letter Z sets bit 0. Bit 35–26 are always returned as zero.

4.8.2. Program Control Table Retrieval (PCT$)

Purpose:

Makes all or specified portions of the information stored in the Program Control Table (PCT) available to the requesting program.

Format:

Two formats are available:

To transfer a maximum of 1000_8 words starting at word zero of the main block:

```
L   A0,(n,buffer-addr)
ER  PCT$
```

To transfer all or part of the PCT starting at any PCT-relative address:

```
L    A0,(0,buffer-addr)
L    A1,(n,relative-addr)
ER   PCT$
```

Parameters:

buffer-addr	Address within the program where the PCT is to be transferred.
n	Number of words to be transferred.
relative-addr	Address relative to the start of the PCT main block from which the transfer should occur.

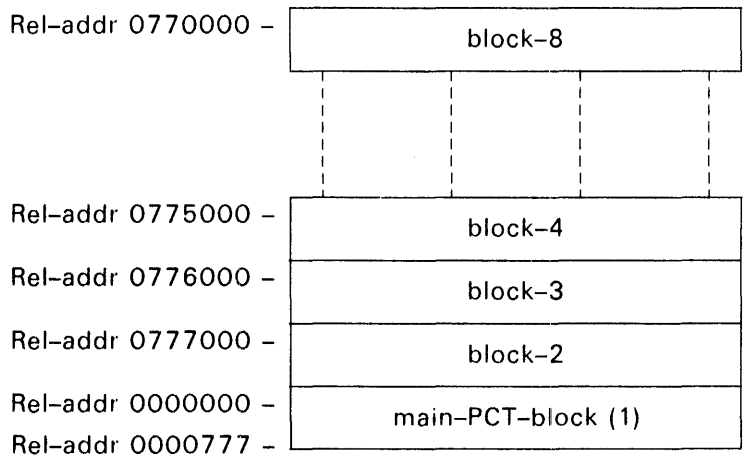
Description:

For information concerning the contents and internal format of the PCT, refer to the Software Release Documentation (SRD). This ER must be used with caution in that Sperry Univac reserves the right to change the content or format of the PCT without notice.

An attempt to transfer information outside of the program's PCT causes the activity to be placed in error mode with a type 4, error 2 status.

A PCT's size is determined by program requirements with a configurable maximum of 42 main storage blocks (512 words each). The structure and addressing of the PCT is illustrated by the following examples.

Assume an eight-block PCT:




```
L,U  A0,BUFFER
L    A1,(02000,0777000)
ER   PCT$
```

This call transfers block-2 and the main-PCT-block, in that order, to the address BUFFER.

```
L,U  A0,BUFFER
L    A1,(0500,0100)
ER   PCT$
```

Transfers 0500 words to BUFFER, starting at relative address 0100 in the main-PCT-block.

```
L,U  A0,BUFFER
L    A1,(0100,0777400)
ER   PCT$
```

Transfers 0100 words to BUFFER from relative address 0400 through 0477 of PCT-block-2.

4.8.3. Master Configuration Table Retrieval (MCT$)

Purpose:

To retrieve information from the Master Configuration Table (MCT) or to read/update the application program area in the MCT. This is a special application ER which is useful only to certain specialized programs. As a result, only those programs which are allowed real-time status are permitted to execute an ER MCT$ which specifies the application area of the MCT. The contents of the MCT are subject to change as new Executive requirements are defined. The current contents and format of the MCT are described in the Software Release Documentation (SRD).

Format:

```
L,U  A0,pktaddr
ER   MCT$
```

4.8.3.1. Packet and Call

MCT information may be retrieved or updated by use of ER MCT$. ER MCT$ requires that register A0 contain the address of a 3-word packet of the following general format:

00	status	unused	function code	read buffer address
01	buffer size			MCT relative address
02	name list	equipment class	number of entries	write buffer address

Word 0

status — Returned by ER MCT$ (see 4.8.3.4).

function code — See 4.8.3.2 and 4.8.3.3.

buffer read address — Address of read buffer. Used only for read functions.

Word 1

buffer size — Read or write buffer size in words.

MCT relative address — Word address relative to start of MCT. Used only for function code 3.

Word 2

name list — Number of words per entry for name list entries.

equipment class — MCT equipment class found in logical name index entry.

number of entries — Number of entries in input list; functions 5, 7, and 8.

write buffer address — Address of write buffer used for write functions or address of name list.

4.8.3.2. Read Functions

The following function codes (decimal) are defined for reading MCT information:

Code	Function
0	Read MCT Pointer Table into buffer provided by the caller.
1	Read MCT Application Area into a buffer provided by the caller. Read specified number of words beginning at specified MCT relative address into a buffer provided by caller. *NOTE:* *This function code reads the MCT excluding the Application Area. If the request includes the Application Area, an error code of 4 is returned.*
5	Read the specified number of MCT logical name index entries into a buffer provided by caller.*
6	Read MCT logical name index entries of a given equipment class code into a buffer provided by the caller.
7	Read a specified number of MCT List Entries into a buffer provided by caller. See code 5.
8	Same as code 5 except equipment class code must be given in S2 of word 2 of packet, and name list entry word 0 contains a Transition Unit position word.
10	Read the MCT System ID Table into a buffer provided by the caller.
11	Read the MCT Time and Date Area into a buffer provided by the caller.
12	Read the MCT I/O Section into a buffer provided by the caller.

Code	Function
13	Read MCT BCA (Bootstrap Communication Area) into a buffer provided by the caller.
14	Read the MCT XER History Entry Table into a buffer provided by caller.
15	Read the MCT Tape Translator Mnemonic/Index Conversion Table into a buffer provided by the caller.
16	Read the MCT Mass Storage ID Table Section into a buffer provided by the caller.
17	Read MCT Console Configuration Section into a buffer provided by the caller.
18	Read MCT Equipment Mnemonic Table into a buffer provided by the caller.

* A name list containing the logical names of the desired logical name index entries is provided by the caller. The name list has entries of 2 or more words (as specified in S1 of packet word 2) with the logical name appearing in word 0 of each entry. In the event an entry's logical name can not be found in MCT, a 1 is returned in S1 of word 1 of the entry in addition to a status of 32 in the packet status cell.

For EXEC callers only, an extra word is supplied following each logical name index entry in the caller's buffer. In H2 of this word is the address of the logical name index entry relative to the start of the I/O section of the MCT.

For all read functions, the read buffer size must be given in H1 of word 1 of ER MCT$ 3-word packet.

4.8.3.3. Write Functions

The following functions are defined for writing MCT information. Only EXEC activities may use the functions with exception of writing of Application Area (59).

Code	Function
2	Write the contents of a caller buffer into the Application Area of MCT.
40	Write the specified number of LNIEs in caller buffer back to their respective addresses in the MCT. Each LNIE in caller buffer must be followed by its address relative to the start of the MCT I/O section.
41	Write the specified number of List Entries in caller buffer back to their respective addresses in MCT. Each List Entry is caller buffer must be followed by its address relative to the start of the MCT I/O section.
50	Write contents from caller provided buffer to MCT System ID Table.
51	Write contents from caller provided buffer to MCT Time and Date Area.
52	Write contents from caller provided buffer to MCT I/O Section.
53	Write contents from caller provided buffer to MCT Bootstrap Communications area.
54	Write contents from caller provided buffer to MCT XER History Entry Table.
55	Write contents from caller provided buffer to Tape Translator Mnemonic/Index Conversion Table.

Code	Function
56	Write contents from caller provided buffer to MCT Mass Storage ID Table.
57	Write contents from caller provided buffer to MCT Console Configuration Section.
58	Write contents from caller provided buffer to MCT Equipment Mnemonic Table.
59	Write contents from caller provided buffer to Audit Trail Table.
60	Write contents from caller provided buffer to MCT Application Area.

For write functions, the write buffer size must be given in H1 of word 1 of ER MCT$ 3-word packet.

4.8.3.4. Error Codes

Error conditions are reported to ER MCT$ callers in S1 of Word 0 of the caller provided 3-word packet:

Code	Function
1	The caller provided read buffer information failed a storage limits check.
2	The caller has provided an illegal function code.
3	The caller is attempting to read/write MCT Application Area without being allowed real-time status.
4	The request is partially completed. *NOTE:* *This status may be returned for various reasons; for example, the caller provided read buffer is too small to contain requested information.*
6	The MCT I/O section relative address provided on write of logical name index entry or list entry does not fall within I/O section of MCT or relative MCT address provided on read with offset (function code 3) does not fall within MCT.
7	The caller provided write buffer information failed a storage limits check.
8	The caller provided function code does not exist.
9	An I/O error was encountered on read/write of MCT information. *NOTE:* *Further ER MCT$ requests are ignored until a reboot is performed.*
32	The provided Name List contains input that cannot be found in the MCT. See function code 5.

4.8.4. Bank Descriptor Index Retrieval and Common Bank Reload (BANK$)

4.8.4.1. BDI Retrieval

Purpose:

BANK$ is a multipurpose ER designed to allow the user to obtain varying kinds of information pertaining to a specified bank.

Format:

The BANK$ calling sequence is:

```
L    A0,parameter
ER   BANK$
```

Description:

The various functions and parameter formats for BANK$ are as follows:

1. parameter=length,address

   Retrieves the BDI of the bank containing the program relative address furnished on the call. Also retrieves information concerning write protection, PSR base, I- or D-bank address, and a nonconfigured common bank indicator. The length parameter, when added to the initial address parameter, defines the highest relative address of the area for which the BDI is to be returned. If a zero length is specified, a value of one is assumed. The length cannot be longer than 131071. To handle lengths larger than this, several successive checks must be done. A status is returned in A0 in the following format:

Bits 0-11	The BDI
Bits 12-14	Zero
Bit 15	1 if the bank is dynamic
Bit 16	1 if bank based on B_D of the PSR
Bit 17	1 if EXEC (common) bank
Bit 18	1 if mapped as a D-bank
Bit 19	1 if bank is under utility PSR base (1110, 1100/40, 1100/80 only)
Bit 20	1 if bank is write protected
Bit 21	1 if bank is nonconfigured common bank

   If A0 = 0, the address range passed was totally or partially outside the user's window.

   The algorithm used to determine if the user's address or address range is within limits is quite similar to the hardware check for an in-limits address, the only exception being that the hardware checks one address at a time, while the software checks a series of contiguous addresses. For 1110, 1100/40, D12 of the main PSR is checked to see which PSR is active (D12 = 0 = main PSR, D12 = 1 = utility PSR). After determining the active PSR, the following order of checking against the storage limits is made:

   a. Starting address against active PSR's I-bank SLR

   b. Starting address against active PSR's D-bank SLR

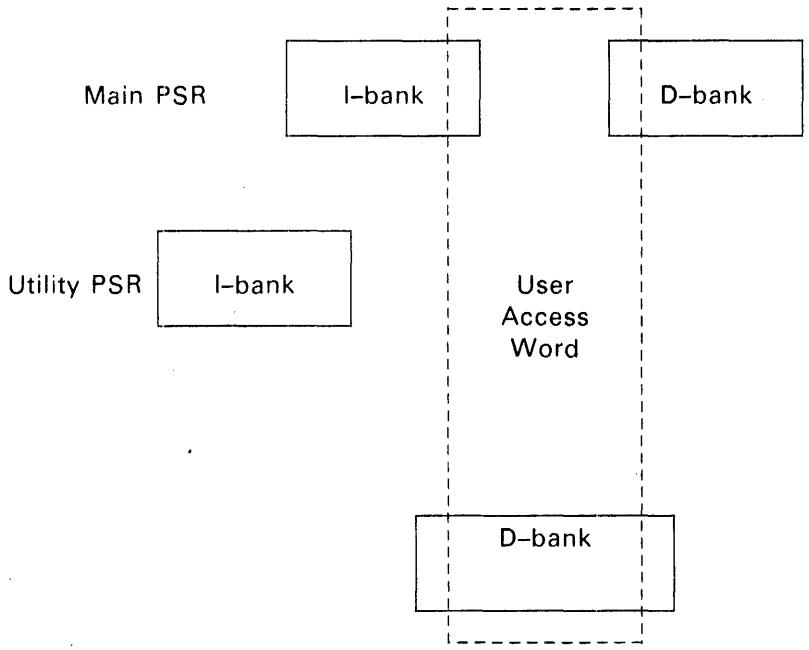
   c. Starting address against inactive PSR's I-bank SLR

d. Starting address against inactive PSR's D-bank SLR

Since a user may have up to four banks (1110, 1100/40, 1100/80 only) active in the program, there exists the possibility that the relative starting address passed on the ER call may exist in more than one bank. However, the first storage limit that includes the relative starting address is used and if the entire address range specified on the call (assuming length ≠ 1) is not within that bank, the check fails and the user's A0 is returned with a value of zero, indicating a failure.

Example:

Assume execution on 1110, 1100/40 and the main PSR (1100/80, BDR) is active and control designators D12 = 0 and D18 = 1.



In this case, the BANK$ request would fail (return A0 = 0) because the main PSR is active and the user access word does not reside wholly in the main I-bank. If the utility PSR was active, however, the request would have passed, as the access word is contained totally within the utility D-bank.

2. parameter,H1 = 1*/17
parameter,H2 = UNUSED

Retrieves BDIs currently active. The BDIs for the main PSR are returned in A0. The BDIs for the utility PSR are returned in A1 (for 1110 and 1100/40/80). Systems). For 1106, 1108, 1100/10/20 Systems, A1 is set to zero. The returned BDIs are formatted as follows:

Bits 0-17	D-bank BDI
Bits 18-35	I-bank BDI

3. parameter = 1*/17+1,BDI

Retrieves storage limits, dynamic/static bit, I-bank/D-bank bit, verified-guaranteed-entry bit, write-protect bit, and guard-mode-protect bit for the bank pointed to by the BDI supplied. The BDI may point to either a common bank or a user program bank. On return to the user, A0

contains the upper storage limits in the left half and the lower storage limits in the right half, expressed as 18-bit word addresses. A1 contains the rest of the information, formatted as follows:

where:

R = dynamic
W = write protect
P = guard mode
V = verified (guaranteed) entry
N = nonconfigured bank
D = D-bank

The remaining bits of A1 are set to zero.

4. parameter = 1*/17+2,BDI

Changes a program bank type from dynamic to static. For this request the caller must have the bank based at the time of the request. If the bank is found to be already static, control is returned to the user with no user action taken. The BDI may not point to a common bank.

5. parameter = 1*/17+3,BDI

Changes a program bank type from static to dynamic. For this request, it is not necessary that the caller have the bank based at the time of the request. If the bank is found to be already dynamic, control is returned to the user with no action taken. The BDI may not point to a common bank.

4.8.4.2. Common Bank Reload

Purpose:

To reload a common bank from LIB$ or a common bank file.

Format:

```
    L    A0,(Reload Type, BDI)
    ER   BANK$
or
    L    A0,(Reload Type, PKT ADDR)
    ER   BANK$
```

where:

Reload Type	See Table 4-2
BDI	Bank Descriptor Index of the bank to be reloaded

PKT ADDR Three word packet as follows:

00	stall time	BDI
01	bank–name	
02	bank–name	

where:

stall time — Length of time (in minutes) allowed for common bank to become available for reload. If time is exceeded, request is ignored and control returned to user.

bank–name — Either bank–name (for S option bank) or element name (see 4.11.1.1 on D option common banks).

Description:

User's A0 is returned with 0 if reload was successful. If the stall time was exceeded, A0 is returned unchanged and control is returned to the user. All other errors causes the user to be placed in error mode (Contingency Type 12, Error Type 10).

The following restrictions apply to usage of the different request types:

1. An immediate request must be issued from the common bank to be reloaded. This restriction does not apply when the bank to be reloaded is a nonconfigured common bank.

2. An activity making an error type request must have the reload bank based.

3. An activity making a stall request must have the bank based or have the common bank file containing the bank assigned with correct keys.

The stall reload request with buffer input (Type = 0400007) allows the user to change the stall time and use a bank name if the reload BDI is not known. A system standard stall time is used if that cell is zero. If the BDI cell in the PKT is nonzero, bank–name is ignored regardless of its value.

Table 4-2. Common Bank Reload Type

Reload Type	Description
0400004	Immediate Reload. Activities having this bank based are not notified that a new copy of the bank has been loaded.
0400005	Error Request. All activities with bank based are put in ERR mode and given a contingency (Type 04) if registered.
0400006	Stall Request With Register Input (First Format). Reload is not started until all tasks currently having access to the common bank terminate.
0400007	Stall Request With Buffer Input (Second Format). This is the only type that may use PKT Address in A0.

4.8.5. Software Instrumentation Package (SIP) Interface (SYSBAL$)

Purpose:

To provide the capability to turn SIP on and off from a user program, and to retrieve the SIP data collected while it was on. In general, all of the SIP options available to the console can be performed via the ER SYSBAL$.

Format:

```
L,U  A0,pktaddr
ER   SYSBAL$
```

Parameters:

pktaddr — Address of SYSBAL$ ER packet

Example:

SYSBAL$ ER Packet

00	status	pkt-length	unused		function code	
01	input data					
02	return data					
03	user-buffer-length			user-buffer-address		
04	time-of-day					

Word 0

status The following status returned:

ER SYSBAL$ Status

Value	Meaning
00	Function completed normally
01	Console has control of SIP
02	Console took control of SIP
03	SIP/Trace feature not configured
04	SIP active from console or other user, cannot set auto mode
010	Console has control of IO Trace in manual mode
011	Console has control of IO Trace and SIP
012	Console has control of IO Trace and took control of SIP
020	Console took control of IO Trace in manual mode
021	Console took control of IO Trace and has control of SIP
022	Console took control of IO Trace and SIP
030	SIP could not be initialized
031	Data requested not available
032	FLGON/FLGOF is not allowed
040	Failed quota bit test/function not allowed
041	Illegal function code
042	Buffer limits out of range
043	Buffer not large enough
044	SIP/Trace not turned on by this user
045	Bad value in SIP request

NOTE:

Nonzero status should be examined by the user to determine how to continue. Negative status indicates a user error.

ER SYSBAL$ Function Codes

Code	Function	Quota Control Group
01	Pass SIP data	1
02	Pass function names	1
03	Pass common bank names	1
05	Pass SIP data and array pointer descriptors	1
06	Pass SIP header block (array pointers, etc.)	1
07	Pass I/O network information	1
010	START	2
020	AUTOFF	2
021	OFF	2
031	STATUS	2
032	FLGON – set collection flag(s)	2
033	FLGOF – clear collection flag(s)	2
040	ON	3
041	AUTO	3
042	IOMAN	3
044	RUNON	3
045	DATON	3
046	IOT	3
050	IOFF	3
052	RUNOFF	3
053	DATOFF	3
054	NOIOT	3
060	TIME	3
062	NOFREE	3
063	FREE	3

NOTE:

If Quota Level 4 is configured, the functions in each Quota control group can be performed by a particular user only if the proper bit has been set by the Installation Manager in the User's Quota Set Description Words. If Quota Level 4 is not configured, there is no such restriction.

Word 1

input data

Word 2

return data

User input for FLGON and FLGOF functions is in the user-buffer. User-buffer-length indicates the number of flags to be changed. Flag mnemonics in Fieldata, left-justified in the user buffer, indicate which flags to change.

Mnemonic	Data
ALL	All data collection
MIS	Miscellaneous; runs open, in core, interrupts, ERs
IO	I/O activity
MEM	Memory utilization, DA activity
CAU	CAU utilization
RSP	DA, Demand Response
FAC	Facility usage
FNC	Function loader activity
EXP	EXPOOL usage
COM	Communications activity
QTM	Dispatcher quantum

Flags may not be changed during a SIP collection. A status of 032 is returned in the user packet if a flag change is requested while SIP is on.

Flag values set by a user do not remain in effect after the user turns SIP OFF (or it is turned off from the console). Values in effect before the user FLGON-FLGOF functions are restored when SIP is turned off.

SIP status for function 031 (STATUS) is returned in words 1 and 2 of the user packet.

SIP Status Return

01	collection flags			
02	unused	SIP/trace status	trace on/off	output time internal (second granularity)

Word 1

Bit	collection flag
0	MIS
1	IO
2	MEM
3	CAU
4	RSP
5	FAC
6	FNC
7	EXP
8	COM
9	QTM
10-35	unused

Word 2

S1 not used

S2 SIP and trace on/off status bits

001 = SIP is on in the START mode
002 = SIP is on in the AUTO mode
004 = SIP turned on by a user
010 = I/O trace turned on by a user

S3 0 = I/O trace is off
1 = I/O trace is on

4.8.6. Jump Stack Editing (EDJS$)

Purpose:

To retrieve the jump history which is captured by the system upon the occurrence of a hardware fault interrupt.

Format:

```
L    A0,(length,buffer-addr)
ER   EDJS$
```

Parameters:

length — The number of jumps to be edited. A default value of 4 is assumed if the specified length is zero and a maximum length of eight is accepted.

buffer addr — Address within the program where the jump history is to be transferred. This area must be a minimum of the length of the number of jumps to be edited. If a length of zero is specified, the minimum length of the area must be four. (A maximum of four jumps may be captured on the 1110, 1100/40 System, and eight on the 1100/80 System.)

Description:

The Executive returns the active BDI and program relative address for each jump in H1 and H2 of each word of the buffer respectively. If the absolute jump address is not within program limits, a zero is returned (this is done since Executive jumps are captured as well as user jumps). Chronologically, word 0 contains the earliest and word 3 the latest jump. Since the jump history is captured on hardware fault interrupts only (guard mode, illegal operation, divide fault, undefined sequence, breakpoint, CAU storage, and GRS parity errors), the ER is useful only if the user has recovered from the fault via a contingency. The ER is legal for programs running on 1106, 1108, 1100/10/20 Systems, but zeros are always returned since no jump history hardware exists for these machines. The Executive limits the number of outstanding captures/edits of the jumpstack per program to five. If multiple recoveries from hardware fault contingencies are done with no subsequent ERs EDJS$, any jump stack captures which would exceed this Executive imposed limit are discarded.

NOTE:

ER EDJS$ does not remove a contingency routine from contingency mode.

4.8.7. Miscellaneous Information Retrieval (INFO$)

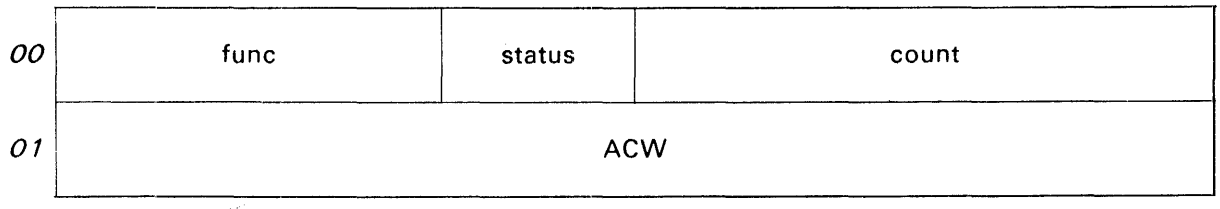
Purpose:

To retrieve system, run, and program oriented parameters in a manner independent of system level.

Format:

```
L    A0,(length,table address)
ER   INFO$
```

Length is the total number of words in the string of INFO$ functions to be processed. Table address is the starting address of the string of INFO$ functions to be processed. For those functions that require a preceding INFILE$ function, one INFILE$ function suffices for the string of functions dependent on the preceding INFILE$. Functions pertaining to a different file require another INFILE$ function.



where:

func — is the functional code mnemonic identifying the information to be retrieved. Function codes are:

Mnemonic	Code Value	Description
USRID$	1	The ACW must specify a 2-word area to hold the user-id of the active run.
SITID$	2	The ACW must specify a 1-word area to hold the site-id of the input device.
INFILE$	3	The ACW must specify a 2-word area that contains a filename. This filename is used on following retrieval functions.
FFILEX$	4	The ACW must specify a 1-word area to hold the number of files extended. An INFILE$ function specifying a tape filename must have preceded.
FBLKSX$	5	The ACW must specify a 1-word area to hold the number of blocks extended. An INFILE$ function specifying a tape filename must have preceded.
FREELX$	6	The ACW must specify a 1-word area to hold the current reel index. An INFILE$ function specifying a tape filename must have preceded.
SUP$	7	The ACW must specify a 1- to 7-word area to hold the current SUP values. The values returned are, in order: current SUPs used, maximum SUPs (from @RUN statement), current CBSUPs used, current CPU SUPs used, current I/O SUPs used, current ER/control card SUPs used, and current voluntary delay time. All values are in 200 microsecond increments.
FEQP$	8	The ACW must specify a 1-word area to hold the specific mnemonic. An INFILE$ function specifying a filename must have preceded.
LNAME$	9	The ACW must specify a 2-word area to hold the logical name(s) of the device(s). For tape files, this is the last unit allocated on. If only one device exists, word two is zero. An INFILE$ function specifying a filename must have preceded.
MODE$	10	The ACW must specify a 3-word area to hold the following tape related information: (An INFILE$ function specifying a tape filename must have preceded.) Data Transfer Format WORD 0 S6 BIT SETTINGS — DESCRIPTION 5 — 1 = 9-track tape 4 — parity (1=even, 0=odd) 3 — 8-bit packed 2 — 6-bit packed 1 — quarter-word 0 — data converter (1=ON; 0=OFF)

Mnemonic	Code Value	Description
		DENSITY WORD 0 S5 (9-TRACK) VALUE — BPI 3 — 6250 BPI 2 — 1600 BPI 1 — 800 BPI (7-TRACK) VALUE — BPI 3 — 800 BPI 2 — 556 BPI 1 — 200 BPI TRANSLATOR INDEX WORD 0 S4 PROCESSOR CODE WORD 1 Defines the format of the data in core. It is identical to the processor code which is used on the @ASG statement for tape translation. TAPE CODE WORD 2 Defines the format of the data on tape. It is identical to the tape code which is used on the @ASG statement for tape translation.
EQUIP$	11	The ACW must specify a 1-word area to hold the equipment index (T1) and the bit settings indicating the type of file or peripheral being interrogated (T3). T2 is presently zero. An INFILE$ function specifying a filename must have preceded. WORD 0 T3 BIT SETTINGS — DESCRIPTION 6 — 1=word-addressable file 1=9-track (if bit 0 also set) 5 — arbitrary device 4 — not used; content zero 3 — FASTRAND 2 — drum 1 — disk 0 — tape
MCSTAT$	12	The ACW must specify a 3-word area to hold the central complex component status. The format of the central complex component status words is illustrated following this table. Each bit of a field represents one component. Each bit which is set indicates that the component is online to the application. This does not necessarily indicate that the component is up and in use in the system. For example bit 0 word 0 represents CPU0. Any field or component which is not meaningful

Mnemonic	Code Value	Description
		or present on the system on which the ER is executed is set to zero. A storage module is defined as a piece of storage that can be partitioned into or removed from an application. For 1106, 1108, 1100/10/20 Systems, storage module 1 will be represented by bit 0; module 2 by bit 1; module 3 by bit 2; and module 4 by bit 3 of word 1. For 1110 and 1100/40 Systems primary storage modules are indicated by bits 0–7 of word 1, and extended storage modules are indicated by bits 8–15 of word 1. For the 1100/80 System without the 4x4 capability, word 1 of the status returned has a bit set for each storage module assigned to SIU upper, and word 2 has a bit set for each storage module assigned to SIU lower. In the SIU field for the 1100/80 system without the 4x4 capability, SIU upper is represented by bit 16, and SIU lower is represented by bit 12. For the 1100/80 with 4x4 capability, word 1 of the status returned has a bit set for each MSU bank that is online in the application.

00	application number	0	SIU components	IOU (IOAU) components	CPU (CAU) components
01	storage components (storage components SIU upper 1100/80 2x2 only)				
02	storage components SIU lower—1100/80 2x2 only				

status is the status returned may contain the following values:

000 Normal completion.

001 Length is odd. There is no ACW for the last function.

002 Illegal function. Code is not defined for this site.

003 Function not available for this configuration.

004 ACW specifies an area which is at least partly outside the user program area.

005 ACW specifies a write protected bank.

006 ACW does not specify enough space for this function. As much data as can be accommodated is transferred. The value in count reflects the number of words required to transfer all the data.

007 ACW specifies too much space. This data is transferred normally. The value in count reflects the actual amount of data transferred.

010 Incorrect input function/data. For INFILE$ function, this error means the file is not assigned to the run.

011 No previous input function given.

012 Transition Unit is offline for the 1100/80 systems. All three of the component status words are returned zero. This status applies only to the MCSTAT$ function.

count is the actual number of words transferred except as described for status 006.

ACW is the number of words and buffer address of deposit area.

4.8.8. Create User-formatted Log Entry (LOG$)

Purpose:

To create a type 32 (user-formatted) log entry in the SYSTEM LOG FILE. The number of user-created log entries (type 32 and type 1) is limited to the value of the CONFIG parameter MAXLOG.

Format:

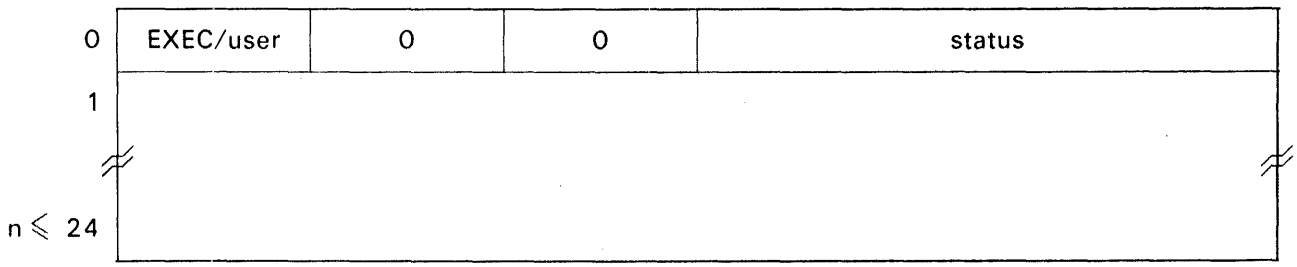
```
L    A0, (length, address)
ER   LOG$
```

Parameters:

Length number of words in table.

address address of run control log entries.

Table has the format:

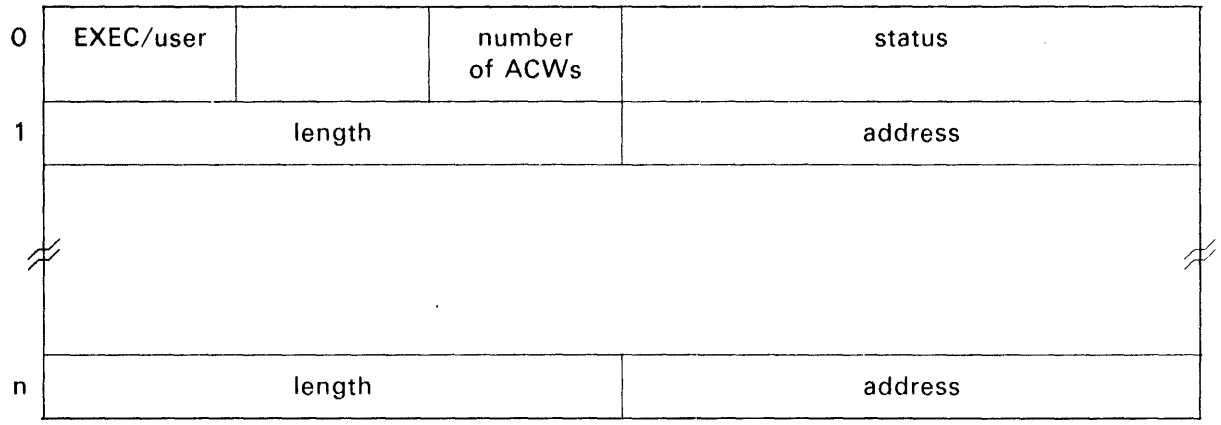


where:

EXEC/user Nonzero indicates that the log entry is to be put on the user chain. Zero indicates that the log entry is to be put on the EXEC chain.

status
0 normal
2 packet is too small to create a log entry

ACW table format:



where:

number of ACWs — The number of log entries to be made.

length — The length of the log entry.

address — The address of the log entry.

4.9. CONTINGENCIES

4.9.1. Introduction

A contingency is an abnormal condition, often associated with an interrupt, which may occur during execution of a program. Typical examples are illegal operation, unsolicited console input, and error mode.

The Executive allows a program to register routines to process contingencies, and transfers control to the appropriate routine should any contingency occur. In the absence of such registration via the IALL$ or CREG$ request (see 4.9.3), the Executive provides a system standard action for each contingency type.

It should be clearly understood that in almost every case a contingency action involves diversion of the execution path of an existing activity, rather than creation of a new activity to handle the contingency. Also, the diverted activity is normally the one to which the contingency specifically pertains.

4.9.2. Contingency Types and Standard Action

Contingencies are classified into a number of different contingency types. These, with their associated standard action are listed in Table 4–3. Contingency types 2, 10, 12, 15, and 16 may be further broken down into error types (see Table 4–4). In addition, in the case of error mode (see 4.1.4), error types are broken down into many error codes, which are given in Appendix C.

The mnemonics listed in Tables 4-3 and 4-4 are standard abbreviations that appear in various system diagnostic messages. They are not tags which may be referenced by a program.

4.9.2.1. Error Termination Considerations

When an activity error terminates, it does not necessarily mean immediate termination of all activities of a multiactivity program, although in practice a program is usually unable to proceed much further when it loses an activity in an error situation. Of course, any activity termination in a single activity program does mean immediate program termination.

Activity error terminations produce a diagnostic message in the run's print file defining the error, the point at which it occurred, and identification of the activity (name or number). In batch mode, a complete control register dump is provided.

When one or more activities of a program error terminate, bits are set in the run condition word (see 3.10.4) indicating this fact, and the run is marked in error. When the program ultimately terminates, further runstream processing is normally limited to processing a postmortem dump, provided a @PMD control statement (see Volume 3-3.21) is the next control statement. A @PMD,E control statement (dump only if an error occurs) is honored in this case (but not after a normal program termination). This run is terminated after PMD processing is completed, unless an @SETC,I control statement (continue in spite of errors - see 3.10.4.1) is in effect.

Table 4-3. Contingency Types

Contingency Type (Octal)	Mnemonic	Description	Standard Action
1	IOPR	Illegal operation (machine instruction undefined)	Error termination of offending activity
2	IGDM	Guard mode or undefined sequence fault	Error termination of offending activity
3	IFOF	Floating-point overflow	Clear A,A+1 registers to zero except for the following instructions: FCL: Clear A only DFP: Clear A+1,A+2
4	IFUF	Floating-point underflow	See number 3, IFOF
5	IDOF	Divide fault (divide overflow)	See number 3, IFOF
6	IRST	Restart	See checkpoint/restart (Section 11)
7	IABT	Abort with contingency	Program and run termination (see 4.9.4.4)
10	IINT	Console interrupt (see also Table 4-4)	Onsite Keyin: Operator message II IGNORED-IN CNTRL MODE if user program has no activities. Operator message NO II CNTGCY FOR RUNID if user program has activities but has not

Table 4-3. Contingency Types (continued)

Contingency Type (Octal)	Mnemonic	Description	Standard Action
			registered for the II contingency. No response if the contingency is given to the user program. Remote Break Key (@@X C): If the program is in control mode (no activities) or has not registered to process the contingency, the contingency is ignored. However, if the 'T' parameter was also present on the @@X command, the current execution is terminated.
11	ITS	Test and Set (TS) instruction interrupt (real-time only)	Control returned to TS instruction (see 10.4.3)
12	ERR$	Error mode (see also Table 4-4)	Error termination of offending activity
13	—	Inter-Activity Interrupt (see 4.3.3.5)	Contingency is ignored.
14	BRKPT	Breakpoint Interrupt (1110, 1100/40/80)	Error termination of offending activity
15	—	CAU/storage or GRS parity error	Error termination of offending activity
16	—	MAX SUPs or PCT overflow	Error termination of offending activity
17	—	Abnormal Program Termination (Common Data Bank-CDB) only	Termination of activity

NOTES:

1. *Contingency types 1 through 5, 14 and 15 are hardware detected. They are discussed in 1108 Multiprocessor System Processor and Storage Programmer Reference, UP-4053 (current version), 1100/10/20 Processor and Storage Programmer Reference, UP-8129 (current version), 1110 and 1100/40 Processor and Storage Programmer Reference, UP-7970 (current version), and 1100/80 System Processor and Storage Programmer Reference, UP-8492 (current version) or UP-8604 (current version). Test and Set (TS) instruction operation is also hardware oriented (see 10.4).*

2. *Error termination is discussed in 4.9.2.1.*

3. *On the 1106, 1108, 1100/10/20 Systems, arithmetic fault (types 3, 4, 5) A-register clearing on standard action is done by examining the a-field of the offending instruction. No clearing occurs if an Execute Remote (EX) instruction was used to execute the offending arithmetic instruction.*

   On the 1100/80 Systems, value of designator register (1110, 1100/40 Systems, PSR' bit D20 determines the action taken for arithmetic faults.

D20 Value *Standard Action*

1 *Action is fully compatible with the 1108 System.*

0 *Interrupt never occurs. The appropriate arithmetic result registers are cleared and instruction execution proceeds in line. Register clearing is done if the offending instruction occurred via Execute (EX) instruction.*

When the user program registers contingency routines to handle one or more of the arithmetic faults (types 3, 4, 5), D20 must equal 1 in order for the interrupt to occur on the 1110, 1100/40, 1100/80.

See Volume 3-2.2.13 for Arithmetic Fault mode of absolute elements.

4. *For 1110, 1100/40, 1100/80 operating systems on the occurrence of a guard mode, illegal operation, breakpoint, undefined sequence (1110 and 1100/40 only), addressing exception (1100/80 only), immediate storage check interrupt (1100/80 only), CAU storage or GRS parity error (1110 and 1100/40 only), the Jump History Stack is captured and saved in a dedicated area of the user's Program Control Table. The user may examine this history via ER EDJS$.*

5. *The Abnormal Program Termination contingency (type 17_8) is available only to Common Data Banks (see 4.9.6 and 4.11).*

Table 4-4. Error Types

Error Name	Error Type (Octal)	Error Code (Octal)	Mnemonic
Contingency Type 2.			
NOTE:			
For 1110, 1100/40, 1100/80 Systems, the following error codes are returned. For 1106, 1108, 1100/10/20 Systems the error code is zero.			
Privileged Instruction Violation	*0*	*1*	
Storage Limits/Write Protect Violation	*0*	*2*	
Control Register Violation	*0*	*3*	
Interrupt Lockout Violation	*0*	*4*	
E-Bit Violation	*0*	*6*	
Table Length Violation	*0*	*7*	
Entry Point Violation	*0*	*10*	
Use Count Overflow on New BD (1100/80 only)	*0*	*11*	
Use Count Underflow on Old BD(1100/80 only)	*0*	*11*	
Contingency Type 7.			
ER ABORT$	0	1	
ER EABT$	0	2	
Real-time program storage check	0	3	
Data corruption storage check	0	4	
Corrupted bank swap-in	0	5	

Table 4-4. Error Types (continued)

Error Name	Error Type (Octal)	Error Code (Octal)	Mnemonic
Contingency Type 10.			
II Onsite Keyin	1		II
Remote BREAK Key	2		RBK
Contingency Type 12.			
I/O Call Error	1		I/O
Symbiont Call Error	2		SYMB
ERR$ Call (ER ERR$)	3		ERR$
Invalid or Bad ER	4		ER
Console Call Error	5		CONS
Communications Error	6		COMM
Communications Error	7		COMM
Common Bank Error	10		REP
Contingency Type 15.			
CAU/Storage Parity Check	0	1	
General Register Stack Parity Check	0	2	
Processor Fault (1100/80)	0	3	
Delayed Storage Check (OLM)	0	3	
Corrupted bank based	0	4	

Contingency Type 16.

The error type and code fields are combined, and subdivided as follows:

```
X XX XXXXXXXXX
A B  C
```

Field A, if set, indicates that the contingency is asynchronous. Used only for common data bank contingencies.

Field B indicates the type of overflow condition encountered, as follows:

- 01 = PCT has reached its configuration - defined maximum sized.
- 10 = PCT for a real-time - program could not be expanded without swapping the program.

Field C indicates the type of request which encountered the PCT overflow (values in octal):

- 001 - PCT space could not be obtained to process an asynchronous contingency. A1 information pertaining to the contingency has been lost.
- 002 - PCT overflowed while attempting to process a FORK$ request.
- 003 - PCT overflowed while attempting to create an interrupt activity for an IOI$ or IOW$ request.
- 004 - PCT overflowed while attempting to process a NAME$ request.

4.9.3. Contingency Registration

4.9.3.1. IALL$

Purpose:

To register a routine to handle one or more contingency types, either for the entire program or for just the requesting activity.

Format:

```
L    A1,(extended-mask)          Required only if registering contingency types above 14(8)
L    A0,(contingency-parameter)
ER   IALL$
```

Description:

The format of contingency-parameter is:

selection-mask	contingency-application	contingency-routine-address

where:

selection-mask

The mask must be set to 0 for ESI contingencies and is ignored if the contingency-routine-addr (below) is zero. All contingency types apply when ESI contingency is registered (see 4.9.5). For program or activity contingencies (see below), a setting of 0 in this field indicates that the information is provided in register A1. If registering contingencies above 14_8, this field must be zero, and the entire selection mask must be supplied in A1.

Only privileged users, i.e., SYS$*DLOC$ assigned to run, are allowed to register for contingency type 15_8.

The bit settings for masks in A0 and A1, and the corresponding contingencies are given below. Unused bits in A1 must be zero.

Contingency Type (Octal)	BIT Set A0	BIT Set A1	Contingency
1	24	0	Illegal Operation
2	25	1	Guard Mode, Undefined Sequence, or Addressing Exception
3	26	2	Floating-Point Overflow
4	27	3	Floating-Point Underflow
5	28	4	Divide Fault
6	29	5	Restart
7	30	6	Abort

10	31	7	Console Keyin
11	32	8	Test and Set (Real-Time Only)
12	33	9	Error Mode
13	34	10	Inter-Activity Interrupt (see 4.3.3.5)
14	35	11	Breakpoint Interrupt (1110, 1100/40, 1100/80 only)
15	-	12	CPU/Storage or GRS Parity
16	-	13	Max SUPs or PCT overflow
17	-	14	Abnormal Program Termination (Common Data Bank only)

contingency-application — Specifies the scope of the registration and may take on the following values:

0 - *Program.* The contingency routine being registered is to apply to all activities of the program (except ESI contingencies).

1 - *Activity.* The contingency routine is to apply only to contingencies pertaining to the requesting activity. In a particular contingency situation, if an applicable routine (including selection mask setting) is registered for both the offending activity and the program, then the activity routine is selected. Otherwise, the program contingency routine is selected, or if it is also not applicable, standard action occurs. However, see CDB contingencies (number 4 and 4.9.6).

2 - *ESI.* The contingency routine is to apply to all contingencies during ESI completion processing for the program. ESI contingencies are handled somewhat differently from non-ESI contingencies (see 4.9.5).

4 - *Common Data Bank (CDB).* The contingency routine is to apply to all (registered) contingencies which occur while executing within a Common Bank. This has higher priority than Activity and Program contingencies if it applies. See 4.9.6 and 4.11 for further discussion of CDBs and CDB contingencies.

040 - *Allow ERs in contingency mode.* The contingency routine is to remain in contingency mode when any of the following ERs are executed: COM$, FORK$, TFORK$, READ$, PRINT$, TWAIT$, APRINT$, TREAD$, CTSQ$, CTS$, CTSA$, SNAP$, AREAD$, ATREAD$, EDJS$, SYMB$, LOG$, and ERRPR$.

contingency-routine-addr — Specifies the address of the first word of the contingency routine. A zero address clears all contingency registration for the application specified in the contingency-application field. (See 4.9.4.)

A program may have as many different contingency routines registered at one time as there are activities, plus a program contingency routine, plus an ESI contingency routine.

The same address may be registered as the contingency routine address for different applications or activities (the difference being the selection-mask settings). This does not include ESI contingencies.

A contingency registration completely cancels any previous registration for the same application (just for the same activity for activity applications).

Console interrupt contingencies (type 010) are by nature not associated with any particular activity and must be registered as program contingencies (application 0). The Executive may divert any activity to process these contingencies, if registered. To avoid an arbitrary activity diversion to process the II/BREAK keyins (type 010), it may be preferable to use the II$ request (see 4.6.2).

The operator keyins X and E cause actions similar to the ABORT$ and EABT$ requests (see 4.3.2.3 and 4.3.2.4, respectively). Program/Activity contingency handlers may not register for these keyins. Common data banks may register for E keyin via Abnormal Termination Contingency (4.9.6.4) and TRMRG$ (4.9.6.5), but not for X keyin.

When registering CDB contingencies (application 4), the contingency routine address must be within a currently-based Common Data Bank, though the IALL$ call may come from any currently-based bank. A Program/Activity contingency routine address may reside in either a CDB or a program bank. An attempt to register contingencies with the contingency routine address not in an acceptable bank results in error termination of the activity (type 4, error 2), without registering the requested contingencies.

The Abnormal Program Termination contingency (type 17_8) may only be registered for CDBs. The bit is stripped from the mask on Program/Activity registrations.

Maximum SUP contingency (type 016) is not associated with any particular activity of a program and must be registered as a program contingency to be honored (application 0).

4.9.3.2. CREG$

Purpose:

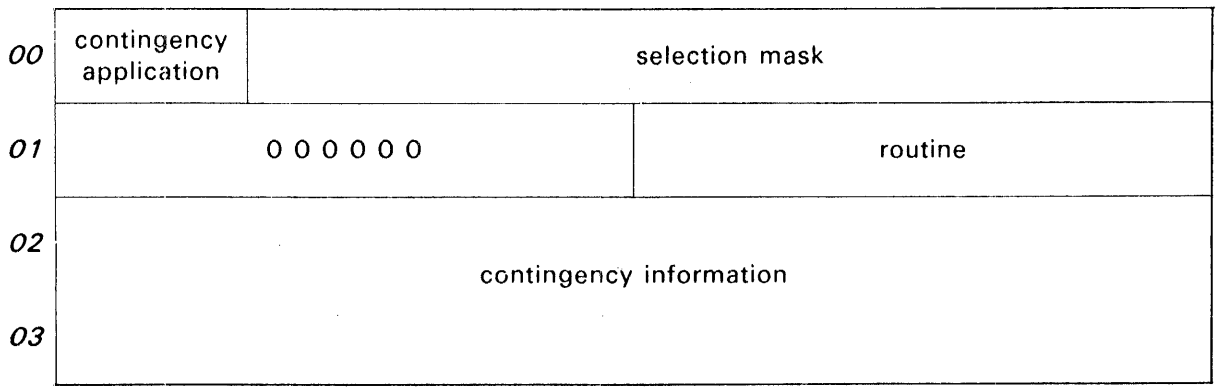
To register a routine to handle one or more contingency types for the entire program, with the routine separate from the contingency packet.

Format:

```
L,U     A0, packet address
ER      CREG$
```

Description:

The format of the packet is:



contingency application	Same as IALL$ (see 4.9.3.1). *NOTE:* *Separate packet and routine allowed for program contingency only.*
selection mask	Same as bit settings for masks in A1 for IALL$ (see 4.9.3.1).
routine	Specifies the address of the contingency routine. If 0, deregistration will occur.
contingency information	Same as the contingency packet for IALL$ (see 4.9.4.1).

4.9.4. Contingency Processing (Non-ESI)

4.9.4.1. Contingency Routine

The first two words of the contingency routine are used as a contingency packet, in which the Executive stores status information concerning the contingency, prior to giving the offending activity control, to process the contingency, at the third word. If CREG$ was used to register for contingency processing, the routine and the packet locations are independent of each other. The format is:

00	error-type	error-code	contingency type	error-address
01			status bits	ER-packet-address-or-H1-of-status-word-for-undefined-sequence
02	(first instruction of the contingency routine)			

where:

Word 0

error-type	See Table 4-4.
error-code	See Appendix C.
contingency-type	See Table 4-3.
error-addr	The address of the offending instruction; or in the case of asynchronous contingencies (see 4.9.4.4), the address of the last instruction executed prior to diverting to the contingency routine. Thus, if it is desired to return to the original execution path after the contingency is processed, the error-addr must be increased by one for use as a reentry address. The error address may not be meaningful for guard mode errors on any hardware other than the 1100/80 System because the hardware does not guarantee a valid interrupt address.

ER-packet-addr — The address of the ER packet associated with the offending instruction. Applicable only for I/O and console error types (see Table 4-4). Also contains H1 of the status word if an undefined sequence interrupt occurred (1110, 1100/40). All of word 1 may also be used on non-I/O error mode contingencies to contain information applicable to the particular error. This is indicated in the sections describing the applicable ERs.

Status-bits — Applicable only for Guard Mode and Undefined Sequence interrupts on 1110, 1100/40, and 1100/80 Systems.

A contingency routine is entered with all control registers for the offending activity loaded as they were when the contingency occurred. Preservation of these registers, if they are needed, is the responsibility of the contingency routine.

Program and activity contingencies are processed serially for the entire program. While an activity is executing a program or activity contingency routine, it is considered to be in contingency mode and no other contingency processing is allowed to occur (except for Common Data Bank contingencies – see 4.9.6). For this reason, contingency processing should be kept as short as possible.

When the activity has completed processing a contingency, it must notify the Executive so that other contingencies may be processed. This is accomplished by executing any ER except those that do not remove contingency mode (see 4.9.2.1). The CEND$ and CRTN$ requests (see 4.9.4.2 and 4.9.4.3) are provided especially for this purpose. Note, that once contingency mode is terminated, the contingency packet is vulnerable to overstoring as the result of another contingency.

4.9.4.2. Contingency Mode Termination (CEND$)

Purpose:

To notify the Executive that the requesting activity has completed contingency processing.

Format:

```
ER  CEND$
```

Description:

Although most ERs (see 4.9.2.1) can terminate contingency mode, the CEND$ request is designed for that specified purpose, and is the most efficient method unless the service provided by another ER is needed. Also see CRTN$ (4.9.4.3).

4.9.4.3. Contingency Mode Termination and Return (CRTN$)

Purpose:

To notify the Executive that contingency processing is complete, and simultaneously returns the calling activity to the address specified in the contingency packet.

Format:

```
ER  CRTN$
```

Description:

Removes the interrupted activity from contingency mode, and returns control at the address (not address+1) specified in H2 of word 0 of the contingency packet. This allows the user activity to process a contingency and return to the original execution path (or to any other point in the program) without destroying a register on the return and avoiding the risk of another contingency interrupt overlaying the packet between a CEND$, jump-indirect-on-the-packet sequence. If user activity is not in contingency mode, the return is to the ER+1. When an activity which is not in contingency mode executes this ER, it has no effect.

4.9.4.4. Additional Contingency Considerations

- Nested Contingencies

  A nested contingency occurs when an activity encounters a synchronous contingency from within a contingency handler. (A synchronous contingency is one that occurs at the same point as its cause.) All nested contingencies are given standard action for the particular contingency type, regardless of registration. If the standard action is termination, the activity is removed from contingency mode, terminated, and some other contingency (if any) is processed. Otherwise, the activity is left in contingency mode. ERR$ is considered a nested contingency, and standard error action occurs. ABORT$ and EABT$ cause standard action but if an ABORT contingency routine is registered, it receives control in the standard manner.

- Multiple (Non-nested) Contingencies

  As stated, contingencies are processed serially within a program. If multiple contingencies occur, one is processed and the rest are queued. The queue is ordered by the switching priority of the offending activity, thus assuring real-time activities proper treatment. When an activity terminates contingency mode, the next contingency on the queue is processed.

  It is possible for multiple (non-nested) contingencies to occur for a single activity, due to asynchronous contingencies. (An asynchronous contingency is one which occurs at a point unrelated to its cause, such as a console interrupt or restart, or an error relating to an asynchronous ER - see 4.1.3 and 4.1.4.) In such cases, each contingency is queued individually and the activity is subject to multiple successive diversions to process each contingency serially.

  Determination of the applicable routine (if any) is made when a contingency occurs. Any change in registration while a contingency is queued does not apply to that contingency.

  When an activity terminates for any reason, any contingencies queued for it are discarded.

- Test-and-Set (TS) contingency processing is available only to real-time activities. This type of contingency is particularly prone to interlock situations, and the programmer should use caution to ensure against such problems. Note that a TS conflict within a contingency routine is given standard action, while a TS conflict outside of a contingency routine, when the program is in contingency mode, results in stalling the conflicting activity (queuing the contingency). See 10.4 for more details on real-time TS processing.

- For ABORT$ or EABT$ requests (see 4.3.2.3 and 4.3.2.4, respectively), the contingency is not honored until all activities of the program have terminated. The user then regains control at the contingency routine with a new activity.

- In some cases, an ER or other system request (e.g., processing an asynchronous contingency) can not be completed because PCT space is not available. This can occur if the PCT has been expanded to its configuration-defined limit, or if the program is real-time and the PCT cannot

be expanded without swapping the program. In those cases where the Executive cannot complete the request via some alternative action (e.g., making do with the PCT space already obtained), the activity in question is considered in error, and is given a contingency (or, in cases where it is more appropriate to do so, is given a bad status, see 4.9.6.3). Typically, this contingency is type 16_8, although some areas have their own error codes for this situation. See Table 4-4 for a summary.

4.9.4.5. Abort Contingency

The abort contingency can report five types of errors. Error codes 1 and 2 result from ABORT$ and EABT$ ERs. See 4.3.2.3 and 4.3.2.4 for discussion of these two aborts.

The abort contingency with error code 3 is given to real-time programs in the event of storage related faults that require the Executive to acquire the program's storage. The appropriate response to an abort with error code 3 set, is to terminate real-time status. The real-time program is terminated without contingency if it fails to respond to the original abort contingency.

The abort contingency with error code 4 indicates the occurrence of a storage offline delayed storage check with the WAN bit set (1100/80). This indicates that the storage area is physically unavailable to the CPU.

Error code 5 on an abort indicates that a program bank containing corrupted data has been swapped into main storage. This condition can arise when a storage fault exists in a program bank's storage, the bank is swapped, and an I/O error has occurred on swapping the bank. The I/O error results in corrupting the content of the bank, and an abort occurs whenever that bank is next swapped into main storage. ER BANK$ can be used to determine the status of program banks and also to remove the indication that a bank is corrupted.

4.9.4.6. Hardware Fault Contingency

Contingency type 015 reports hardware faults. Error code 1 reports CAU/storage parity check for the 1110,1100/40 Systems, or immediate storage checks on the 1100/80 System. Error code 2 reports General Register Stack parity errors. Online maintenance programs receive an error code 3 for delayed storage checks. Online maintenance must register for program-wide contingency in order to properly capture storage check status words. Error code 4 reports the basing of a corrupted program bank. The status word contains the BDI of the corrupted bank. ER BANK$ can be used to determine the status of the bank and also to remove the indication that a bank is corrupted.

4.9.5. ESI Contingencies

The communications handler provides the user with the capability of processing various contingency conditions that might occur while executing an ESI activity. To establish an ESI contingency, the user real-time program must register the contingency via the IALL$ and CREG$ requests (see 4.9.3). All contingency types that can occur within an ESI activity are processed by the specified ESI contingency routine.

The format of the ESI contingency packet is:

00	error-type	error-code	contingency-type	error-addr
01	TS-indicator	not used	status bits	ER-packet-addr or H1 of status word for undefined sequence
02	(first instruction of the contingency routine)			

Word 0

error-type — Error-type is 07 for communications

error-code — The error-codes are:

- 60_8 – Indicates contingency type 1 through 5 and 11
- 61_8 – ESI ACT$ or ADACT$ request error
- 62_8 – ESI CADD$ or ADACT$ request error
- 63_8 – Invalid ER request
- 64_8 – ESI time-out

contingency-type — The contingency-type codes are:

- 1_8 – Invalid operation
- 2_8 – Guard mode
- 3_8 – Floating-point overflow
- 4_8 – Floating-point underflow
- 5_8 – Divide fault
- 11_8 – Test and Set ESI contingency error
- 12_8 – Error mode (see error-code). Applicable only to $61_8 - 63_8$
- 14_8 – Breakpoint (1110, 1100/40, 1100/80)
- 15_8 – CPU/Storage or GRS parity (1110, 1100/40, 1100/80)
- 16_8 – MAX SUPs or PCT overflow

error-addr — Same as for non-ESI (see 4.9.4.1) except that for error mode contingencies (type 12), the error address is the true reentry (no incrementation is needed).

Word 1

TS-indicator — Set equal to 1 by the Executive prior to giving control to the contingency routine.

ER-packet-addr — The address of the ER packet associated with the offending instruction. Applicable only to contingency type 012 (see Tables 4-3 and 4-4). Also contains H1 of the status word if an undefined sequence interrupt.

Status-bits — Applicable for guard mode and undefined sequence.

The following contingency type and status values can be received:

Contingency Type	Status Bits Value	Reason
2	00	Unmodified Guard Mode
	01	Privileged instruction violation
	02	Storage limits/write protect violation
	03	Control register violation
	04	Interrupt lockout violation
	05	(Unused)
	06	E-bit violation
	07	Table length violation
	010	Entry point violation
	011	Use count overflow/underflow
15	01	CAU Storage parity
	02	CAU GRS parity

As for non-ESI contingencies, ESI contingencies are initiated serially. Prior to terminating contingency mode, however, the routine may enable contingency processing on another CPU by clearing the TS indicator (S1, word 1) in the contingency packet. This indicator serves to protect the packet contents from being overwritten until the contingency routine has had a chance to retrieve the information pertinent to the contingency. Termination of the ESI contingency mode also enables other contingency processing.

Should another contingency occur while in contingency mode, the ESI activity is terminated and the communication line associated with that activity is deactivated for I/O operations. The following message is issued to the console and the master run-log as an error message to indicate the terminating condition:

run-id line-id ESI TERMINATION *nnmm*

The line-id field of the message indicates the line associated with the termination, and the error field gives the error code. The first reference to the deactivated terminal by the real-time program causes a non-ESI contingency for the referencing activity with an error type 07 and an error code 010. The *nnmm* field of the message specifies an error type and error code as indicated above.

To terminate ESI contingency mode normally, only those Executive Requests specified for normal ESI activities may be used: EXIT$, ACT$, CADD$, and ADACT$. Any reference other than those indicated above result in a contingency within a contingency, causing terminal deactivation as described in the preceding paragraphs.

ESI contingencies are independent of non-ESI contingencies. A program may process ESI and non-ESI contingencies concurrently.

4.9.6. Common Data Bank (CDB) Contingencies

Due to their unique ability to be associated with several runs simultaneously, Common Data Banks (CDBs) are provided with the ability to process contingencies independently of any attached programs, as well as some special privileges not available to regular program banks. In this section, the term 'common bank' means any Executive bank (E-bit set in BDI), whether or not write-protected. The term Common Data Bank (CDB) refers to a write-enabled common bank. Program bank means any noncommon bank (E-bit not set in BDI).

4.9.6.1. CDB Contingency Registration

CDB contingencies may be registered in two ways: as a configuration option when the bank is defined at system generation time, or via ER IALL$ or CREG$.

- If the bank is configured to process contingencies, registration occurs automatically for the bank the first time it is loaded. The bank must have a Collector-defined entry point, and the locations at entry-point+1, and entry-point+2 must contain the following data:

ep + 1	0	contingency packet address
ep + 2	0	contingency mask

  The packet address and contingency mask must be as discussed under IALL$ (4.9.3), and the packet must lie within the CDB. It is suggested that the instruction at the bank's entry point be a jump around these two data words. System generation registration is effective only when the bank is loaded from the program file. Changes to these data words after the bank has been loaded have no effect on contingency registration.

- CDB contingencies may also be registered or changed dynamically via IALL$ and CREG$, as described in 4.9.3. Note that the packet must lie within a currently-based CDB, and the application field must contain a 4. The ER IALL$ or CREG$ may be done from any currently-based bank. CREG$ may be used to allow a write protected common bank to register for contingency processing by specifying a packet in a write enabled common bank.

4.9.6.2. CDB Contingency Processing

When a contingency occurs for a user activity, the following algorithm determines who is to process it:

- Contingencies which occur while executing within a program bank are routed to the activity or program contingency handlers, or are given standard action, regardless of whether a CDB is currently based.

- If a contingency occurs while executing within a common bank, the active addressing window is searched for a currently-based CDB which has registered to process that contingency. The order of the search is:

  1. The bank in which the activity was executing
  2. The opposite bank in the same PSR under which the activity was executing
  3. (If 1110, 1100/40, 1100/80), the I-bank of the opposite PSR
  4. (If 1110, 1100/40, 1100/80), the D-bank of the opposite PSR.

- If no CDB was found to process the contingency, control is routed to the activity, program, or standard action handler.

  If a CDB was found to process the contingency (only the first find is considered, even if several currently-based CDBs have registered to process the contingency), the packet's limit is checked (if the packet is found to lie outside the CDB, or if the CDB is a D-bank which is overlapped by the I-bank of that PSR, the activity is unconditionally terminated in error); the packet is initialized; and control is returned to the contingency routine. Contingencies for a CDB are processed serially, with multiple contingencies being queued until able to proceed. Exit from the contingency routine is done in the standard manner. (See 4.9.4.2 and 4.9.4.3.)

  When setting contingency mode for a CDB, D12 of the main PSR is set to the correct PSR.

Common data bank contingencies and program/activity contingencies are processed independently, and may run in parallel for different activities of the same program. However, a single activity may not be in both types of contingency mode at once. If an activity should enter a common bank without exiting program contingency mode, then encounter another contingency within the common bank, the program/activity contingency mode is terminated before the CDB contingency handler is given control. At this time, the next contingency (if any) queued for the program is activated. Nested contingencies in either common banks or in programs are given standard action. A contingency is considered nested if it is synchronous, and occurs in a common bank while already in CDB contingency mode, or in a program bank while already in program/activity contingency mode.

For CDBs only, asynchronous contingencies (IO$, II/BK, INT$, RL) are flagged by setting bit 35 in the first word of the contingency packet. Thus, the actual error type, error code, contingency type information must be obtained by ANDing out (or equivalent) the high-order bit in H1 of the first word of the packet.

NOTE:

Test and Set and queuing an activity is an illegal instruction in CDB contingency mode processing.

4.9.6.3. Queuing Contingencies for the User Program (CQUE$)

Purpose:

To queue a contingency for an activity from a common bank, to be processed upon return to the calling program.

Format:

```
L    A0,(error type, error code, contingency type)
L    A1,(auxiliary information)
ER   CQUE$
```

Description:

The (error type, error code, contingency type) parameter is the same format as supplied in the contingency packet (see 4.9.4.1). Auxiliary information is any 36-bit quantity, which appears in the second word of the contingency packet when the contingency is processed.

The information in A0 and A1 is saved, and processed as a contingency after control is returned to the user program, whether via LJB/LIJ/LDJ, or by a straight jump. The contingency is not necessarily processed immediately upon return to the program bank, and the activity may execute for some time before it is diverted. The feature is intended mainly as a means of passing asynchronous

contingencies (IO$, INT$, II/BK, RL), which occur in the common bank, on to the calling program. If the program does not have the particular contingency registered, it is given standard action.

CQUE$ does not remove the activity from contingency mode, so A0, A1 may be restored and control returned to the interrupted code via CRTN$ (see 4.9.4.3).

Upon return from CQUE$, one of the following status codes is left in H1 of A0:

000000 – normal completion

400000 – request rejected because of undefined contingency type.

400001 – request rejected because a PCT buffer could not be obtained to save the information.

Several CQUE$ requests may be made for the same activity, but may not necessarily be processed in the order they were queued.

CQUE$ may be done from a program bank, but, since the activity is already executing in a program bank, it is subject to immediate diversion to process the contingency just queued.

Contingency types 7 (abort) and 6 (RSTRT) may not be passed via CQUE$, nor may any undefined contingency types.

4.9.6.4. Abnormal Program Termination while Executing within Common Banks

When an activity is abnormally terminated (aborted) while executing within a common bank, an Abnormal Program Termination contingency (type 17_8) is generated. This contingency is to notify a CDB that the activity has been aborted, and applies to all abort conditions (ABORT$, MAX TIME, E keyin, etc.), except the X keyin. It is the responsibility of the CDB to see that the activity is eventually terminated.

The preferred way is via ER EXIT$. Alternatively, the CDB can queue the contingency via CQUE$ (see 4.9.6.3), and eventually return to the caller, at which time the activity is terminated. However, if the CQUE$ request returns an abnormal status, the activity must be terminated via ER EXIT$ or ER ERR$.

If the activity is not executing in a common bank when the termination request occurs, or if no active CDB has it registered, the Abnormal Termination Contingency is given standard action, which is immediate termination of the activity.

Processing for the Abnormal Termination Contingency (ATC) should consist mainly of immediate cleanup tasks (clearing locks and closing up queues, for example) necessary to the integrity of the Common Bank system. For more extensive cleanup under more general termination circumstances, the ER TRMRG$ (see 4.9.6.5) is provided.

When a program is aborted, activities are awakened from the following wait states:

- DACT$/II$
- TSQ and C$TSQ waits
- Tape I/O requests waiting for the tape to be mounted.
- TWAIT$

In all cases involving ERs (all above except TSQ wait), the address given in the abnormal termination contingency packet points to the ER. Note that in the DACT$/II$ case, this differs from the address returned when a DACT$/II$ activity is awakened for other asynchronous contingencies. (ER address −1 is returned in such cases.) Abnormal termination contingency is not normally considered a recovery situation, but is intended mainly as notice to the CDB that the activity has been aborted. Thus, the CDB would not typically want to put the activity back in the wait state, without at least checking things out. This is particularly important for the C$TSQ wait, where the ER cannot be reexecuted without first setting the Test and Set. By pointing to the ER, the standard contingency return of reentry +1 returns control following the ER.

If the CDB's logic requires that return be made to the interrupted code (in general, this is the case), the logic following the DACT$/II$ and C$TSQ must allow for the fact that activation may occur via the abnormal termination contingency, as well as via the normal sequence of events. This can generally be handled by the appropriate use of flags. (S3 of the TSQ word is available in the C$TSQ case.) If it is necessary to reenter a DACT$/II$ wait, no problems should be encountered, as the reactivation indicator is set if an ACT$ has occurred while the activity in question was processing the contingency. However, if a C$TSQ must be reentered, the possibility that a C$TSA was performed and lost while the activity was processing the contingency must be considered. This can also be resolved by appropriate use of flags.

When an activity is activated from a TSQ wait (is on T/S queue for T/S failure rather than C$TSQ), to process an abnormal termination contingency, the reentry address points to the T/S instruction −1.

In the case of the tape I/O request not being performed, the activity is errored with an I/O 032 error with a 032 status placed in the I/O packet. In most cases this contingency is processed before the abnormal termination contingency. The CDB contingency handler should be prepared for this case if it is doing tape I/O.

The CDB is given as much time as needed to process the Abnormal Termination Contingency, hence much care should be taken to avoid infinite loops. The X keyin must be used to terminate such loops.

4.9.6.5. Notification of Common Banks of Activity Termination (TRMRG$)

Purpose:

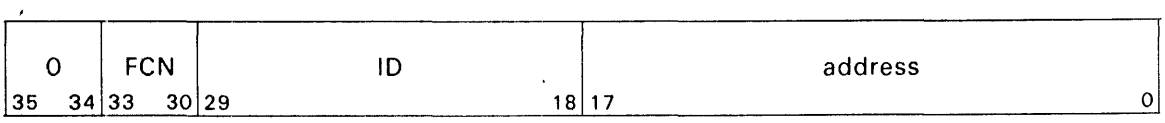
To register a common bank to be notified when an activity (task) terminates.

Format:

```
L    A0,(parameter)
ER   TRMRG$
```

Description:

The parameter word has the following format:



where:

address — is the relative address of an entry point within the calling bank where control is to be sent at activity termination time.

ID — is any 12-bit value supplied by the caller. The low order 12 bits of the caller's BDI is suggested.

FCN — indicates the type of registration to be performed;

0 = Activity registration
1 = Program registration
2 = Activity deregistration
3 = Program deregistration

TRMRG$ is provided so that common banks may receive control, at activity or program termination time, to perform any necessary cleanup operations (releasing space allocated to this activity or program, for example). TRMRG$ differs from the Abnormal Termination Contingency (see 4.9.6.4) in several ways:

- TRMRG$ applies to all common banks, whether or not write-enabled, and whether or not contingency processing is registered.
- TRMRG$ applies whether or not the common bank is based at the time the activity (program) terminates.
- TRMRG$ applies to all types of termination, normal and abnormal, with the exception of the X keyin.

For TRMRG$ registration (FCN = 0 or 1), the ID and address, along with the caller's BDI, are saved and associated with the activity or program. When the activity (program) terminates, each registered common bank, in turn, is based for that activity (or the last activity to terminate for program termination) and given control at the registered entry address, with the ID in A0. Upon receiving control, the registered common bank is the only active bank for the activity (all other banks are unbased - the common bank must LBJ/LIJ/LDJ to any other banks needed). Its position in the addressing window (main/utility I-bank/D-bank) will be the same as it was at registration.

The activity has whatever register set (major/minor), name, ID at the time of termination, except for the program termination case, where it has no name or ID and has a minor register set. TRMRG$ does not go through the contingency handler and there is no logical lockout or queuing; i.e., the processing routine must do its own lockout or be reentrant. When through, the processing routine must terminate via ER EXIT$, at which time the next registered bank is processed. When all banks have been given control, the activity (program) is finally terminated.

The ID given on the call is used for identification purposes, to detect multiple registrations for the same activity/program, and for deregistration. A multiple registration is one with the same ID, for the same activity/program, from the same common bank as a previous registration, and is ignored. Deregistration (FCN = 2 or 3) causes a previous registration (ID, BDI, Activity/Program combination) to be deleted.

Upon return from the ER TRMRG$, a status code is left in bits 35-33 of A0 as follows:

0 = normal completion

1 = registration request already registered, or deregistration request already deregistered.

4 = registration not performed because PCT core space critical.

The following points concerning TRMRG$ should be noted:

- TRMRG$ must be done separately by each common bank for each activity/program of interest.
- TRMRG$ may be called only from common banks. Attempts to register from a program bank results in error termination of the activity, with a code of type 04, error 71.
- Banks using TRMRG$ should be extremely careful to avoid infinite loops, since the only way to terminate TRMRG$ mode (unless done voluntarily) is via the X keyin. The X keyin causes immediate termination of the run, with no regard for CDB contingencies, TRMRG$ registrations, or anything else.
- @PMD after processing TRMRG$ registrations may be meaningless, since the banks which were active at initial program termination have been unbased by the time the PMD is taken.
- Program TRMRG$ requests are processed after the ABORT$ contingency (if any) is given. However, if the last activity in the program to terminate went through activity TRMRG$ processing, the ABORT$ contingency will guard mode because the program banks have been unbased.
- Activity TRMRG$ registration is not copied to the new activity on a FORK$ request.

4.10. MISCELLANEOUS EXECUTIVE REQUESTS

4.10.1. Dynamic Request of Control Statements

The following ERs can be initiated during program execution.

4.10.1.1. Request with Fieldata Images (CSF$)

Purpose:

Permits the user program to submit certain control statements in Fieldata for interpretation and processing during program execution rather than from the runstream.

Format:

```
L    A0,(image-length,image-addr)
ER   CSF$
```

Parameters:

image-length	Length in words of the control statement image (maximum 40_{10})
image-addr	Address of the buffer that contains the image

Description:

The submitted image must be in the identical Fieldata format, including the @ as the first (left most) character of word 0, that it would have been if it had been submitted as a regular control statement in the input run stream.

Termination of scan results from whichever of the following occurs first: a comment of blank–period–blank is encountered, a blank following the last allowable parameter field is encountered, or the image–length in H1 of register A0 has been reached.

Maximum allowed value for image–length is 40_{10} words; 14_{10} is assumed if 0 is given.

The control statements which may be processed by the CSF$ request are:

@ADD	@CKPT	@QUAL
@ASG	@FREE	@RSTRT
@BRKPT	@LOG	@START
@CAT	@MODE	@SYM
		@USE

Control statement syntax and other errors generally result in error mode termination with contingency type 12_8 (see 4.9.4), error type 4, and one of the error codes 40_8 through 44_8 as described in Appendix C.3.

When certain control statements are submitted by the CSF$ request, register A0 is returned containing status or error information. For the facility request statements (@ASG, @CAT, @FREE, @MODE, @QUAL, and @USE), bits set in register A0 upon return from the CSF$ request indicate that either the request was rejected or that it was accepted with precautionary warnings (see Appendix C.2 for interpretation of bit settings in A0). The meaning of the bits set in register A0 upon return from processing the @BRKPT and @SYM symbiont control statements are described in Appendix C.4.2. The status codes returned in register A0 for an @CKPT CSF$ request are described in Appendix C.5. On return from @CKPT request, H1 of register A0 contains the checkpoint number.

On return from a CSF$ @START request, register A0 contains codes that are described in Appendix C.4.

For CSF$ requests for processing @LOG and @RSTRT control statements, no status information is returned in register A0.

Example:

The following example illustrates how an @ASG control statement (3.7.1) is submitted by an Executive Request to the CSF$ function.

Assume the user wants to assign a temporary sector–formatted scratch file named FILEA and to reserve two tracks. This can be coded as follows:

```
                L,U     A0,IADD
                ER      CSF$
                •
                •
                •
IADD            '@ASG,T   FILEA,F/2  .  '
```

The blank–period–blank sequence terminates the image scan.

4.10.1.2. Request with ASCII Image (ACSF$)

Purpose:

Permits the user program to submit control statements in ASCII for interpretation and processing during program execution rather than from the runstream.

Format:

```
L    A0,(image-length,image-addr)
ER   ACSF$
```

Description:

The interpretation of parameters is identical to that for the CSF$ request (4.10.1.1). The ACSF$ request is similar to the CSF$ request. Maximum allowed value for the image-length is 60_{10} words; 21_{10} is assumed if 0 is given.

4.10.2. Retrieving Information from and Altering the Processor State Register

Purpose:

The SPD instruction and the Executive Request PSR$ allow the retrieval of certain information from the PSR. The LPD instruction and the Executive Request PSR$ allow an activity to dynamically set or clear certain bits within the processor state register. Although the LPD and SPD instructions are not part of 1106 and 1108 hardware, certain of their capabilities are provided via simulation on 1106 and 1108 Systems.

The following designator bits are affected by ER PSR$ and by the LPD/SPD instructions:

quarter-word mode	D10 or bit 17 in following discussion
double-precision underflow mode	D5 or bit 32
floating-point compatibility mode	D8 or bit 35

The following designator bits are affected by the LPD/SPD instructions on 1110, 1100/40, 1100/80 Systems only:

character addressing mode	D4
active PSR/SLR indicator/selector	D12
floating-point residue store control	D17
arithmetic exception interrupt control	D20

A description of the operation of the LPD/SPD instructions and of the PSR$ Executive Request follows.

4.10.2.1. Store Processor Designators (SPD)

Format:

```
SPD  u,Xx
```

Description:

PSR D-bits → $(U)_{6-0}$;

D20 → Bit 6
D17 → Bit 5
D12 → Bit 4
D10 → Bit 3
D8 → Bit 2
D5 → Bit 1
D4 → Bit 0

D20, D17, D12, D10, D8, D5, and D4 of the PSR are transferred to bit positions 6 through 0 of U, respectively. Bit positions 17 through 7 of location U are cleared to zero. Bit positions 35 through 18 of location U are left undisturbed if U is a storage location, they are cleared to zero if U points to a register. Index register incrementation and indirect addressing are allowed as for any other instruction.

For 1106/1108 Systems, a simulated SPD always shows D17 and D20 as being set because they are always logically set on these systems. For 1100/10/20, the hardware places zeros in these bits. The SPD instruction is most often used in conjunction with the LPD instruction. (See examples under LPD description.)

4.10.2.2. Load Processor Designators (LPD)

Format:

LPD u,Xx

Description:

$U_{6,5,3-0}$ → PSR (DR on 1100/80)

Bit 6 → D20
Bit 5 → D17
Bit 3 → D10
Bit 2 → D8
Bit 1 → D5
Bit 0 → D4

Bits 6, 5, 3, 2, 1, 0 of U are transferred to the PSR designator bits D20, D17, D10, D8, D5, and D4 respectively. This instruction uses the 18-bit value U, rather than the 36-bit contents of U. Index register incrementation and indirect addressing are allowed as for any other instruction. Note that the absence of a bit in the U-field of the instruction constitutes a directive that the corresponding bit in the PSR be cleared. For 1106/1108 Systems, the LPD instruction must not attempt to clear D17 or D20, or set D4 because these systems do not have the corresponding hardware capabilities (for 1100/10/20, these bits are ignored). As noted above, a simulated SPD always shows D17 and D20 as being set.

Examples:

The following sequence of instruction is a recommended means of setting a designator bit when it is not known whether the bit is currently set:

```
SPD    A2
OR,U   A2,VALUE
LPD    0,A3
```

where VALUE may be any of the following octal values:

```
0001  corresponding to D4
0002  corresponding to D5
0004  corresponding to D8
0010  corresponding to D10
0040  corresponding to D17
0100  corresponding to D20
```

The following sequence is recommended to clear a designator bit when it is not known whether the bit is currently set:

```
SPD      A2
AND,U    A2,-VALUE
LPD      0,A3
```

where VALUE may again be any of the listed values in the previous example.

4.10.2.3. Executive Request PSR$

Purpose:

Allows an activity to dynamically set or clear certain bits within the PSR or allows retrieval of certain information from the PSR.

Format:

```
L    A0,(parameter-word)
ER   PSR$
```

Description:

Bits 0, 2, and 3 in the parameter word control the modification of PSR bits 17, 32, and 35, respectively. When a control bit (0, 2, and 3) is set, the associated bit in the PSR is set to the value of the corresponding bit in the parameter word. For example, if bit 0 in the parameter-word is set and bits 2, 3 are not set, the content of bit 17 of the parameter-word is placed in bit 17 of the PSR, and bits 32, 35 are not disturbed.

Examples:

The following examples illustrate typical parameter words loaded into control register A0 and their interpretations for a PSR$ request.

Parameter Word (Octal)	Description
000000000000	No modification of existing PSR; enables readout of PSR in control register A1.
440000400015	Set bits 17, 32 and 35 of PSR (initiates quarter-word mode, double-precision underflow mode, and floating-point compatibility mode).
000000000015	Clears bits 17, 32, and 35 of PSR.
440000400010	Sets bit 35 of PSR (initiates floating-point compatibility mode); all other modes remain the same (bits 17 and 32 are not interpreted when control bits 0 and 2 are not set).
040000400011	Clears bit 35 of PSR (floating-point compatibility mode); sets bit 17 of the PSR (quarter-word mode); other modes remain the same.

4.10.3. Main Storage Snapshot Dump (SNAP$)

Purpose:

Provides a snapshot dump printout of the contents of selected control registers and program storage as an aid for debugging.

Format:

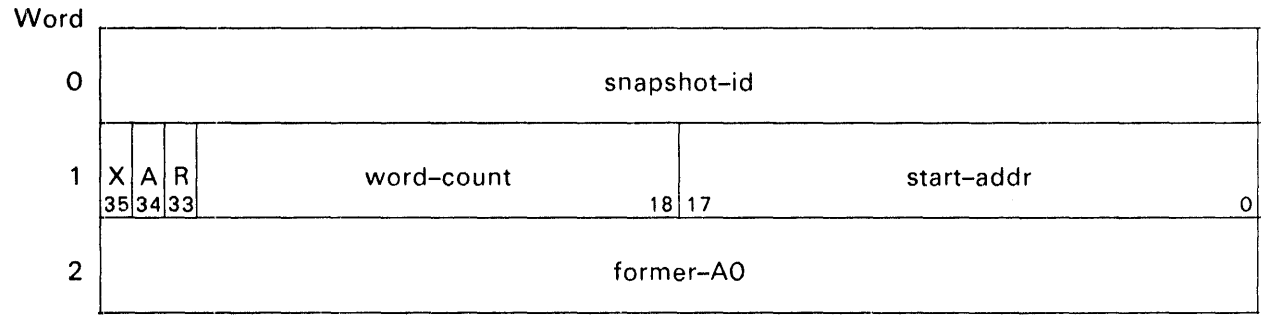
```
S    A0,pktaddr+2
L,U  A0,pktaddr
ER   SNAP$
```

The above three instructions, a J $+4 instruction, and the 3-word packet are generated via the L$SNAP procedure using the following call:

```
L$SNAP   'snapshot-identifier',   xar,word-length,start-address
```

Description:

Pktaddr is the address of a packet whose format is:



where:

snapshot-id	Six-character Fieldata name used to identify the dump.
x,a,r	Used to designate registers to be dumped as follows: 1. If bit 35=1, dump all X-registers. 2. If bit 34=1, dump all A-registers. 3. If bit 33=1, dump all R-registers. If all three bits are equal to 0, no registers are dumped.
word-count	Number of words of main storage to be dumped.
start-addr	Starting program address of the main storage area to be dumped.
former contents of A0	Save area for register A0. This is needed to capture the entire environment. Register A0 is restored using this value before returning control to the program.

Be careful when using the SNAP$ request in a multiactivity program because packet usage is not reentrant.

4.10.4. Setting User Breakpoint (SETBP$)

Purpose:

To allow the user to set the 1110, 1100/40, or 1100/80 Systems' programmable breakpoint register.

Format:

```
L,U  A1,BDI
L    A0,(breakpoint parameter)
ER   SETBP$
```

The equivalent calling sequence may be generated via the S$ETBP procedure using the following call:

S$ETBP breakpoint-addr 'control' address-mask BDI

Description:

The format of the breakpoint parameter is:

where:

control — Bit 30 specifies initiation of a breakpoint interrupt during a store instruction. A W is used to specify this bit if S$ETBP is used.

Bit 31 specifies initiation of a breakpoint interrupt during a read from storage. An R is used to specify this bit if S$ETBP is used. Both bits 30 and 31 (W, R) can be specified simultaneously.

Bit 32 specifies initiation of a breakpoint interrupt for an instruction address comparison. On 1110, 1100/40 Systems, this bit must not be set if either bit 30 (W) or bit 31 (R) is set. A P is used to specify this bit if S$ETBP is used.

Bit 33 specifies that a BDI is supplied in A1. If a BDI is supplied on the S$ETBP call, this bit is automatically set.

Bits 34 and 35 are not used, but must be zero.

addr-mask — Control bits which when set force comparison for corresponding bits 5-0 of the relative breakpoint address. Bit 24 controls bit 0, bit 25 controls bit 1, etc. Any combination of bits can be set. This allows breakpointing on a block of 077 addresses (addr-mask=077). This is meaningful only on 1110, 1100/40 Systems.

breakpoint-addr — The relative address to be compared to either an instruction or storage operand address. Only one comparison can be made at a time.

The BDI (Bank Descriptor Index) supplied in A1 is accepted only when bit 33 is set in the breakpoint parameter. The ability to include a BDI with the SETBP$ request permits setting of a breakpoint to a bank not currently in the user's window. If a BDI is not specified (bit 33 not set), on return from the ER, A1 contains either the previous BDI if a SETBP$ had previously been initiated and was currently active or the BDI reflecting the breakpoint address specified in the breakpoint parameter. If the relative address specified overlaps more than one bank, the BDI is determined from the active PSR.

To clear the breakpoint setting, A0 is set to zero on the ER SETBP$ request.

If bits 30-32 are zero on the ER call, bit 32 (P-bit) is automatically set.

4.10.5. Error Message Printout (ERRPR$)

Purpose:

To print an error message for a user.

Format:

```
L,U   A0,pktaddr
ER    ERRPR$
```

Packet Format 1 (Executive Messages)

00	35	ec	ct	error address
01	0			packet address
02	status (CSF$)			

where:

et — Error Type (see Table 4-5)

ec — Error Code (see Table 4-5)

ct — External Contingency Number (see Table 4-5)

Packet Format 2 (User Messages)

00	0	error number
01	0	
02	0	

where:

Status returned in A0

0	Message found and printed
1	No message found for the given number
2	No message lookup because user PCT is at maximum size
3	No message lookup because the E$ORMSG element is not in SYS$*LIB$
4	No message was printed because an error was detected in the E$ORMSG element or an I/O error occurred while reading the E$ORMSG element

When a status other than zero is returned, the message being looked up has not been printed or not completely printed. For user messages no printout at all may have occurred. For Executive messages the error-id line and other packet information has been printed. The user program initiating the Executive Request must decide when a nonzero status is returned. The error number for user messages is defined by those processors using the facility or by the local site.

Table 4-5. EXEC Codes Currently Assigned

Contingency Number	Error Type	Error Code(s)	Description
1	0	0	illegal operation
2	0	0 1106, 1108, 1100/10/20 1 thru 011 1110, 1100/40/80	unmodified guard mode (see 4.9.5)
3	0	0	floating point overflow
4	0	0	floating point underflow
5	0	0	divide fault
6	0	0	restart
7	0	1-5	ABORT$
010	0	1,2	console interrupt, @@X C
011	0	0	test and set interrupt (real-time)
012	1	0 thru 040	I/O error or status
012	2	2 thru 045	symbiont errors
012	3	0	user did an ER ERR$

Table 4-5. EXEC Codes Currently Assigned (continued)

Contingency Number	Error Type	Error Code(s)	Description
012	4	1 thru 077	error in user Executive request
012	5	0 thru 4	console errors (ER COM$)
012	6	1 thru 050	communications errors
012	7	1 thru 077	communications errors
012	010	1 thru 044	multibank program errors
013	0	0	interactivity interrupt
014	0	0	breakpoint interrupt (1110, 1100/40, 1100/80 Systems only)
015	0	1 thru 4	CAU/Storage, GRS parity errors
016	0	0 thru 077	maximum run time exceeded or PCT maximum size exceeded
0	040	0 thru 047	Facility request status bits
0	043		initial load errors
0	047	00	max time
0	047	01	quota abort
0	047	02	illegal account file reference
0	047	03	AWAIT$-DACT$ ambiguity
0	047	04	operator terminated run with E-keyin.
0	047	05	PCT overflow
0	047	06	real-time program PCT expansion
0	047	011	PCT and program too large for storage
0	047	012	mass storage overflow
0	047	013	I/O error or checksum error loading bank
0	047	014	bad read or write keys on CSF$
0	047	016	disk pack registration error or wrong pack
0	047	017	operator Xed the run

Table 4-5. EXEC Codes Currently Assigned (continued)

Contingency Number	Error Type	Error Code(s)	Description
0	047	020	program aborted due to storage check
0	047	022	ER ABORT$
0	047	023	ER EABT$
0	047	024	program aborted due to storage check
0	047	025	program aborted due to swap-in of a corrupted bank

4.11. DYNAMIC AND COMMON BANK USAGE

The following paragraphs describe programming considerations for dynamic banks and common banks. This is not a complete discussion but provides guidelines for dynamic bank utilization.

4.11.1. Usage of Multiple Banks

One function of multiple banks is to provide for the dynamic loading of additional instruction and data space.

Another use is to provide instructions and data which can be shared, through simultaneous use, by many independent runs.

4.11.1.1. Common Banks

Common banks allow for very fast inter-program communications. They may provide considerable savings in total main storage requirements when commonly used system processors and library subroutines are written as common banks. Several items must be taken into consideration before deciding to write a routine as a common bank.

1. Unless the routine is used frequently enough that the probability is high that more than one run will use the routine each time it is loaded, the additional costs may not be justified. If the common bank is not initially based (see 3.4.4.2.3), it is not loaded as part of the initial program and the use by a single run results in more overhead than if the routine was collected with the program.

2. If the routine is to be called dynamically rather than initially based, is the execution time within the routine long enough to justify the overhead of the call? LBJ/LIJ/LDJ cost in instructions:

	Common-Static	Common-Dynamic	Common-Static Guaranteed-Entry	Common-Dynamic Guaranteed-Entry
1106, 1108, 1100/10/20 LIJ/LDJ/LBJ	150	180	180	190
1110, 1100/40 LIJ/LDJ	1	90	90	100
1110, 1100/40 LBJ	150	180	180	190
1100/80 LIJ/LDJ/LBJ	1	1	1	1

The quoted costs assume the common bank is in storage at the time of the LIJ/LDJ/LBJ. When the common bank is not in storage, a storage allocation and load from mass storage are required. For common-static banks and for highly used common-dynamic banks, the probability the bank is in storage at LIJ/LDJ/LBJ time is very high.

The costs for guaranteed-entry banks on the 1100/80 System assume the bank has been constructed to take advantage of the hardware validity-bit (in the BDW) capability (guaranteed entry point is the first address of the bank). If the bank has not been so constructed, the cost is on the order of 90 instructions.

4.11.1.1.1. Common Bank Reload

A common bank may be reloaded from LIB$ or a common bank file via an ER BANK$ or an unsolicited keyin 'RL BANKNAME'. Both the in-core copy and the swapfile copy of the bank are released, so that the swapfile copy is also renewed if the bank is swapped after the reload.

For reload purposes, common banks are identified by bank-name. (On ER Type requests which give a BDI as input, the BDI is immediately converted to a bank-name.) Thus, common banks must have unique names. For S option configured banks, a REPROG card format has been defined which includes a 'BANKNAME' subfield. That name must match the one used on the bank statement at collection time. For non-S-option configured banks the element name given on the REPROG card is used as the bank-name.

4.11.1.1.2. Non Configured Common Banks

Non Configured Common Banks have been introduced to alleviate many of of the difficulties associated with the development and testing of common banks.

There are two basically different types of Non Configured Common Banks (NCCB). One, called a Stand Alone NCCB, has only one bank within its absolute element. The symbolic BDI used to reference a Stand Alone NCCB is the name of the absolute element which contains the Stand Alone NCCB. The program which uses a Stand Alone NCCB has no special restrictions. The absolute element which contains the Stand Alone NCCB must be in a registered file.

The other type of Non Configured Common Bank is called an Integrated NCCB. A program which uses an Integrated NCCB must be executed from a registered file. The absolute element which is executed contains the actual code or data which is to be used as an Integrated NCCB.

With Non Configured Common Banks, there is no longer a need for a unique BDI value; however, a Non Configured Common bank does require a unique symbolic name.

For this reason, and for reasons of internal resources, it is desirable to restrict generation of Non Configured Common Banks to elements coming from registered files. There is no longer a restriction that Common Banks come from LIB$, though LIB$ is always a registered file for generation of Non Configured Common Banks. Any number of additional files can be configured at system generation time.

The code for a Stand Alone Common Bank is generated by placing an absolute element of the desired name in any of the registered files. This absolute element must contain a single bank. A Stand Alone NCCB is referenced by a program which contains a void bank (length = 0) of the desired name which has the 'S' option on the bank card. This program can be executed from any file. The first reference to a Stand Alone NCCB actually generates the Bank. Subsequent references to the bank use the single system copy of the bank taken from core or loaded from swapfile.

The code for an Integrated NCCB is generated by putting the S option on the bank card of the bank which is an Integrated NCCB. The bank name supplied to the collector for the bank with the S option becomes the name of the Integrated NCCB. This must not be a void bank. The absolute element which contains the Integrated NCCB may contain any number of (1) real program banks, (2) references to Stand Alone NCCBs, (3) references to Configured Common Banks, and (4) other Integrated NCCBs. The name given to an Integrated NCCB must be unique among all Non Configured and Configured Common Banks including Stand Alone NCCBs. An Integrated NCCB defined in this manner is generated on the first execution of this program from any of the registered Non Configured Common Bank files. Subsequent executions of this absolute element uses the single system copy of the Integrated NCCB taken from core or swapfile.

Guaranteed Entry, Test and Set Queuing, or Contingency Registration can be specified for a Non Configured Common Bank by putting the appropriate option on the bank card at collection time. For Stand Alone NCCBs, the collection which generated the single bank absolute element must have specified the appropriate options.

4.11.1.2. Additional Instruction and Data Space

Some programs have periodic or occasional need for additional space. One such class of programs is real-time programs which have periodic or data-dependent requirements for additional space. This additional space can be achieved in several ways.

1. Overlay segments. This is the most economical technique. No additional main storage allocation is required. Existing space is just reused. There are two potential drawbacks to the use of overlay segments. One is that the previous contents of the main storage area are not saved. Another is that overlay usage is nonreentrant since it involves instruction modification.

2. MCORE$ and DSEGs are the next most economical means. Some additional allocation is required. The disadvantage of this technique is that sufficient relative addressing space may not be available. Also, if several such areas must be defined, they must be layed out end-to-end, which uses up additional relative addressing space, or they are nonreentrant. They may not be feasible for real-time programs, as contiguous space for the expansion may not always be available.

3. Additional banks. These provide additional space which need not be contiguous to any of the previously in-use space. Also, in the case of dynamic banks, the space is not allocated until requested. Static banks allow rapid access without having to wait for allocation and loading but require main storage even when not in use. The instructions and data do not overlay other instructions or data, thus allowing full reentrant usage. Additional banks do have some disadvantages. They require a PSR base, which means that for 1106, 1108, 1100/10/20 Systems the original I- or D-bank is no longer visible. The cost of maintaining, allocating, and loading the additional banks is high compared to segmentation. (A bank's contents is never overlayed without being saved first, in case it is requested again.) Each activity of a program may have its own complete set of banks with the same relative address ranges, but different physical space. Thus, a transaction program could have many activities, each of which goes to a common routine to obtain and identify transaction and then acquires, via LBJ/LIJ/LDJ, the correct set of banks to process the transaction.

4.11.2. Write Protect Mode

Any bank may be specified to be write protected (read only). This is especially valuable for common banks, as it protects them from accidental or malicious overwriting. For 1106/1108 Systems, the write protect mode simulation may be disabled when the system is generated. If this is done, access to write protected banks (except the PCT) is still allowed, but without the protection enabled. In write-protect mode, the Executive does not allow stores or I/O to occur into the area. Thus, I/O packets, READ$ buffers, and I/O input buffers may not be in a write protected bank. The size of a write-protected bank may not be changed using MCORE$/LCORE$ (see 4.7).

4.11.3. Inter-Bank Addressing

Any bank may reference space in any other currently visible bank. For program static and dynamic banks this does not present a problem since all the banks are collected together. Some means must be provided for common banks. Several linkages are available which may be used independently or jointly to obtain the desired results.

4.11.3.1. Collection

The V and Y options on the @MAP control statement (see Volume 3-2.2.1) and the V option on the I-bank and D-bank directives (see Volume 3-2.2.2.18) are provided to aid in the formation of common banks. With these options all the banks may be collected together so that all absolute addresses are assigned and then absolute elements can be generated consisting of only one bank. These single bank elements may be entered into the system library as common banks. To generate a common bank using the Collector:

1. Collect all the banks together using the appropriate I-bank and D-bank directives. Local element inclusion (see Volume 3-2.2.5.6.2) may be used to obtain copies of an element in more than one bank. Multiple copies of implicitly included elements do not occur.

2. Use a V option on all but one of the bank directives to generate an element containing only the code for that bank. If necessary, use an ENT directive to specify the starting address for this element. A starting address is necessary for guaranteed entry common banks.

3. Repeat step two for all other single bank elements to be created.

4. Generate the element to be called by the @XQT or processor call statement by using a V option on all the bank directives which were previously used to create separate elements. Use an ENT directive, if necessary, to specify the starting address.

An alternative method for collecting a common bank is to use the S option on bank cards when collecting a program containing common banks. If the bank or banks which are to be common banks have the S option on their I-bank or D-bank statement at collection time, then it is possible to define that bank(s) as a common bank(s) at system generation. This technique has two distinct advantages:

1. Only a single collection is required.

2. The absolute element can be tested in its final form prior to inserting it into the system.

4.11.3.2. Register Basing

A bank need not be collected with another bank in order to reference it. In fact, routines may be written so that they are independent of the overall structure and content of the other bank. In this case, all required addresses must be loaded into registers by the calling routine. Referencing is then through an index register plus offset.

An extension of this technique is to have the calling routine pass one address in a register, with this address being the location of a table (calling sequence) of addresses and values.

4.11.3.3. Collector Produced Tables

The Collector produces tables of program information upon request. The tables, COMMN$, ENTRY$, and XREF$, equate program identifiers to program addresses. The addresses of these tables may be passed to the common bank which uses them to locate data and instructions in the program. (See Volume 3-2.2.8 for the format of these tables.)

4.11.4. Bank Address Limits

No checks are made when a bank is accessed via LIJ/LDJ/LBJ to prevent part or all of the other banks from being hidden (not visible). Normally, if the banks were all collected together, there is not a problem. In any other case this must be considered when constructing the calling program. Either the information to be referenced must be located in the area which remains visible or the address limits must be adjusted. The address limits may be adjusted through the use of the bank directive (see Volume 3-2.2.2.18).

4.11.5. Executive Requests within Common Banks

4.11.5.1. MCORE$ and LCORE$ Usage

The MCORE$ and LCORE$ Executive Requests require that a common bank be based before its size can be changed.

4.11.5.2. Contingency Registration and Processing

Any contingency routine in effect at the time of an LBJ/LIJ/LDJ instruction remains in effect when the new bank is used. The common bank may register for contingency processing independently from the user program, and in certain circumstances is given first chance to process a contingency. (See 4.9 for a complete discussion of contingency handling by common banks.)

4.11.5.3. CMS$ and CPOOL$ Usage

A CMS$ request may be executed from within a common bank but the ESI completion activity does not have access to the common bank. A CPOOL$ request may also be executed from within a common bank, but the buffer space may not be within the common bank.

4.11.5.4. LOAD$ Usage

The LOAD$ request may not be used to load a segment within a common bank. Segments which load only into program static banks and based dynamic banks may be loaded while common banks are based.

4.11.5.5. IO$ Request

The IO$ request may not be used if the packet or I/O buffer is contained within a common bank. An attempt to do so results in a 023 I/O error status if the packet is in a common bank, or in a 025 I/O error status if the I/O buffer is in a common bank.

4.11.6. Dumping Common Banks

The Executive does not provide the capability to obtain postmortem dumps of common banks through the use of the PMD processor. The programmer must provide dumps by using SNAP$ or the XDIAG$ library routines.

4.11.7. Data Protection and Activity Synchronization

4.11.7.1. TS Usage

The TS instruction (and TSS and TCS on 1110, 1100/40, 1100/80 Systems) may be used to protect data within common banks. Data protection in common banks implies data protection between activities of different programs. TS conflicts between batch, demand, and deadline activities are resolved but may involve considerable swapping overhead before conflict is resolved. If real-time activities are involved, correct level control must be used. That is, all activities referencing the TS cell must raise to the same real-time level. Test and Set Queuing (TSQ) should be used if activities at differing priority levels may be involved. This removes the need to utilize level control (see 4.11.7.3).

4.11.7.2. ACT$, DACT$, AWAIT$ Synchronization

The normal activity synchronization requests may only be used between activities of the same program. They can be used within common banks but only for those cases where it is known that all the activities involved are part of the same program.

4.11.7.3. TS Queuing

The TS queuing mechanism (see 4.3.4) provides an effective technique for resolving the data protection and activity synchronization problems. TS queuing registration allows the usage of this feature within the program. Registration for common banks is accomplished by system generation parameters or Collector directives and operates independently of the status of any program. The

usage of the automatic queuing mechanism and the Executive Requests applies to all activities involved, whether or not they are of the same program. Level control for data protection is resolved by having all waiting activities deactivated because they are not candidates for execution or for main storage if the program was swapped.

Interprogram activity synchronization may be achieved by:

1. For each activity involved define a TS cell. This cell is the control cell for the activity and serves the same purpose as an activity name.

2. The equivalent of DACT$ is:

```
        TS      CELL    . PROTECT CELL
        TZ,S3   CELL    . DID "Somebody Activate Me"
        J       AWAKE   . YES
        C$TSQ   CELL    . No, Wait for Activation
        TS      CELL    .
AWAKE   SZ,S3   CELL    . I Am Active, Clear the Flag
        C$TS    CELL    .
```

3. The equivalent of ACT$ is:

```
        L,U     reg,1       .
        TS      CELL        . PROTECT DATA
        S,S3    reg,CELL    . SET 'SAM' Flag in Case It Is Still Active
        C$TSA   CELL        . Wake Up Activity If It Is Waiting
```

Many additional schemes may be invented. Counters or bit masks may be used to program an AWAIT$ capability without requiring Executive defined activity-ids. The "Somebody Activate Me" flag could be changed to a counter and used to cause activation only after a certain number of activation sequences or to cause as many successful passes through the deactivation sequence as there were intervening activation attempts.

6. Input/Output Device Interfaces

6.1. INTRODUCTION

This section describes I/O device interfaces to mass storage, magnetic tape, low speed onsite devices, and special handling of peripheral devices. The communication device interfaces are discussed in Section 9.

The I/O packet structure, Executive Requests (ERs), and procedures which are applicable to both magnetic tape and mass storage devices are presented. These are followed by details relating to specific magnetic tape and mass storage applications. Finally, special device handlers are described.

6.1.1. Basic I/O Executive Request

Magnetic tape and mass storage files are accessed through the packet mode using an Executive Request, and register A0 loaded with the packet address as follows:

```
L,U  A0,pktaddr
ER   entrance-tag
```

Parameters:

pktaddr — Address of the I/O packet (see Figure 6-1). The length of the request packet can vary from four to eight words, depending upon the operation desired.

entrance-tag — The available entrance-tags are as follows:

IO$ (See 6.3.4)

IOI$ (See 6.3.5)

IOW$ (See 6.3.6)

IOWI$ (See 6.3.7)

IOXI$ (See 6.3.8)

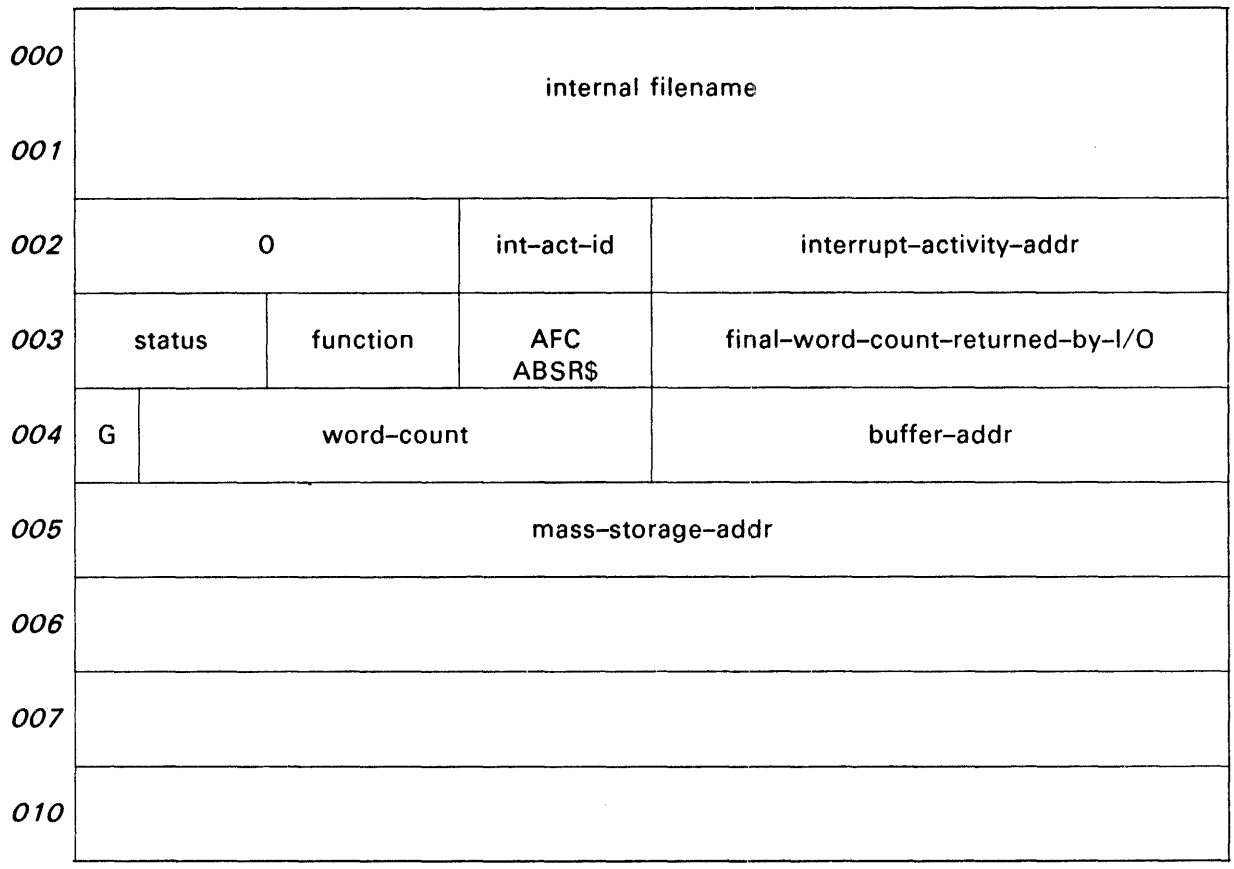


Figure 6-1. I/O Packet, Mass Storage, and Magnetic Tape Peripheral

Words 0 and 1

The internal filename (see 2.6.2) used in all references to the file. It is specified in Fieldata, left justified and space filled.

Word 2

int-act-id — The numeric identity (1-35) used to identify the interrupt activity if synchronization is intended with some other activity. Must be zero if no activity-id is desired (IOI$ and IOWI$ only). For IOXI$ the interrupt activity has the same activity-id as the initiating activity.

interrupt-activity-addr — The address at which the user program receives control upon occurrence of an interrupt signifying completion of the I/O operation (IOI$, IOWI$ and IOXI$ only).

Word 3

status — The status of the last function performed. Bit 35 must be zero when making an I/O request (see 6.1.3).

function — Denotes the function to be performed (see Table 6-1).

AFC — The abnormal frame count value for magnetic tape files only (see 6.4.2.5).

final-word-count-returned-by-I/O — For any function involving data transfer, this field contains the exact number of words read or written. For magnetic tape or the end of a mass storage file, this number may differ from the access word on read operations.

Word 4

An I/O access word; or for GW$, SCR$, and SCRB$ functions, this word contains the number of access words in H1, and the address at which the string of access words begins in H2. For BDW$ and BDR$, this word contains the number of access word pairs in H1 and the address at which the access word pairs begin in H2.

G — Designator (bits 34 and 35) to increment or decrement buffer-addr by 1 for each word transferred.

00_2 - increment
10_2 - decrement
01_2 - no increment or decrement
11_2 - output (write) operations - same as 01_2

input (read) operations - same as 01_2 if configuration parameter SKDATA is set to zero. If SKDATA is set not-zero, the buffer address contained in the packet has no meaning; the data is transferred to a reserved storage location on non-1100/80 Systems or the hardware skip-data feature is utilized on 1100/80 Systems.

word-count — For standard I/O, number of words to transfer; for scatter/gather I/O, the number of access words; for I/O by BDI, the number of access word pairs.

buffer-addr — For standard I/O main storage address at which transfer is to begin; for scatter/gather I/O, the address of the chain of access words; for I/O by BDI, the address of the chain of access word pairs.

CAUTION

If a single mass storage I/O request causes more than one word of data to be transferred to a single storage location, the order in which the data is transferred to that storage location is unpredictable. Two examples of this are: (1) any input (read) ACW with a G field of 01_2 and a word count greater than one, (2) a scatter read (SCR$) function which has two or more ACWs reading data into the same storage location.

Word 5

For mass storage files, this word contains the logical mass storage address at which the described I/O operation is to start. This address is relative to the start of the mass storage file. For sector-formatted mass storage files, the address is the start of a sector, and consecutive addresses are 28 words apart.

Words 6 through 010

These are as defined for the function contained in word 3,S2 (see 6.5.3).

Table 6-1. Octal and Mnemonic I/O Codes Defined in SYS$*RLIB$

Function	Octal	Symbol
Write by BDI	04	BDW$
Write	10	W$
Write end of file	11	WEF$
Skip write	13	SW$
Gather write	15	GW$
Acquire	16	ACQ$
Extended Acquire	17	EACQ$
Read	20	R$
Read backward	21	RB$
Read and release	22	RR$
Release	23	REL$
Block read drum	24	BRD$
Read and lock	25	RDL$
Unlock	26	UNL$
Track search all words	30	TSA$
Track search first word	31	TSF$
Position search all words	32	PSA$
Position search first word	33	PSF$
Search drum	34	SD$
Block search drum	35	BSD$
Search read drum	36	SRD$
Block search read drum	37	BSRD$
Rewind	40	REW$
Rewind with interlock	41	REWI$
Set mode	42	SM$
Scatter read	43	SCR$
Scatter read backward	44	SCRB$
Move forward	50	MF$
Move backward	51	MB$
Forward space file	52	FSF$
Backspace file	53	BSF$
Read by BDI	54	BDR$
Mode set	55	MS$

I/O by BDI access word pairs:

The format for each access word pair presented on an I/O by BDI request is as follows:

G	word-count	buffer-addr
0		BDI

H1 of the second word must be zero.

6.1.2. Interrupt Activity

The interrupt activity is the same as other registered activities using the FORK$ function (see 4.3.1.1) except for the following:

- The priority of the activity is raised to the highest possible level within the program class of the user program; that is, for a batch user program, these I/O completion activities receive control before any other batch program activity.
- The interrupt routine is not interrupted in favor of any other similar activity of the same program. All are queued in a first-in/first-out list of all activities without regard to priority within the class.
- Any Executive Request or the exceeding of a CPU time quantum removes the interrupt activity from the high priority list and returns it to the user program's priority.
- The control register subset in the interrupt routine is limited to registers X8 through X11, A0 through A5, and R1 through R3. Register A0 contains the I/O packet address. When recovery is suppressed (R$, RB$, SCR$, SCRB$ functions only), 011_8 is returned as the I/O packet status. For VIC/VIIIC servos, register A1 of the interrupt activity contains the EI status word.
- In the absence of any other request, the normal program status can be restored by using the UNLCK$ request (see 6.3.9).
- When multiprogramming, every attempt is made to provide proper switching by allowing immediate access to the amount of computation required to initiate a new I/O operation following any other I/O operation. The difficulty lies in preventing abuse of the high priority assigned to interrupt activities. The available facility is limited to initiate a new I/O operation after having checked the status of the previous I/O operation.

6.1.3. Queuing

When an I/O request is made, the Executive sets S1 of word 3 (the status word) equal to 040_8 to indicate I/O in progress. Before setting the word negative, a check is made to determine if the packet's status is already negative, indicating a possible loop. If it is, the activity requesting I/O service is terminated with a error code 27_8 error type 01 status. The I/O status code 027_8 is not stored in the I/O packet. When the request is completed, a positive value is placed in the status word. No housekeeping is necessary by the user; encountering an initial negative value in the packet can be interpreted as a software logic error.

Efficient utilization of all mass storage types, including sector-formatted mass storage, dictates that servicing requests for a given file are not restricted to the order of submission. The nonsequential processing of I/O requests to mass storage results in faster servicing and more efficient utilization of the system's I/O facilities. Testing each packet is necessary to ensure completion. Do not assume completion by testing a subsequent packet.

6.2. I/O PACKET GENERATION

There are two basic procedures for generating I/O packets: I$OT (see 6.2.1) is used to generate I/O packets for magnetic tape files, and I$OD (see 6.2.2) is used for mass storage file I/O packets. An I/O operation with interrupt involves inclusion of additional parameters for word 2 of the I/O packet (see Figure 6-1). The tag on the procedure line is allocated to the first word of the I/O packet.

6.2.1. Magnetic Tape I/O Packet Generation (I$OT)

6.2.1.1. Magnetic Tape I/O Function without Interrupt

Format:

```
I$OT    'filename',function    word-count,buffer-addr [,G]
```

Parameters:

filename	Specifies the internal filename of the tape file being referenced.
function	Symbolic or numeric code identifying the function to be performed (see 6.4 and Table 6-1).
word-count	For standard I/O, the number of words to be transferred; for scatter/gather I/O, the number of access words; for I/O by BDI, the number of access word pairs.
buffer-addr	For standard I/O, the main storage address at which the transfer begins; for scatter/gather I/O, the address of the chain of access words for I/O by BDI, the address of the chain of access word pairs.
G	Increment-decrement function designator as follows: 'D' - Decrement 'N' - Inhibit incrementation or decrementation Omit this parameter for incrementation.

Example:

```
TOP          I$OT    'T101',W$    200,BFR
```

Filename T101 is entered in words 0 and 1 of I/O packet (see Figure 6-1). Octal function code 010 associated with symbolic function code W$ (write) is placed in the S2 portion of word 3. A total of 200 words are to be written (placed in word-count portion of word 4), starting at main storage location BFR. The omission of the G parameter indicates that incrementation is employed.

6.2.1.2. Magnetic Tape I/O Function with Interrupt

Format:

```
I$OT    'filename',function,interrupt-activity-addr,int-act-id  word-count, buffer-addr [,G]
```

Parameters:

filename	Specifies the internal filename of the tape file being referenced.
function	Symbolic or numeric code identifying function to be performed (see 6.4 and Table 6-1).
interrupt-activity-addr	Address of activity to which control is passed after completion of I/O function.
int-act-id	Any integer (1_{10} to 35_{10}) that identifies the interrupt activity if synchronization with another activity is desired.
word-count	For standard I/O, the number of words to be transferred; for scatter/gather I/O, the number of access words; for I/O by BDI, the number of ACW pairs.
buffer-addr	For standard I/O, the main storage address at which the transfer begins; for scatter/gather I/O, the address of the chain of access words; for I/O by BDI, the address of the chain of ACW pairs.
G	Increment-decrement function designator as follows: 'D' – Decrement 'N' – Inhibit incrementation and decrementation Omit this parameter for incrementation.

Example:

```
TOP             I$OT    'AABBCD23',R$,GOTOAD,3 15,LOCR,'D'
```

Filename AABBCD23 is entered in words 0 and 1 of I/O packet (see Figure 6-1). Function code 020 (read) is placed in S2 of word 3. The address GOTOAD is placed in H2 of word 2, while the interrupt-activity-id (3) is placed in S3 of word 2. The number of words (15) to be read into main storage starting at location LOCR is placed in the word-count portion of word 4; LOCR is placed in the buffer-addr portion of word 4, and the 'D' indicates a decrementing function to control the direction of the words transferred.

6.2.2. Mass Storage I/O Packet Generation (I$OD)

6.2.2.1. Mass Storage I/O Function without Interrupt

Purpose:

Creates I/O packet for mass storage files.

Format:

```
I$OD    'filename', function   word-count, buffer-addr, [,G]   drum-addr  [, search-sentinel]
```

Parameters:

filename	Specifies the internal filename of the file being referenced.
function	Symbolic or numeric code identifying the function to be performed (see 6.5, 6.6, and Table 6-1).
word-count	For standard I/O, the number of words to be transferred; for scatter/gather I/O, the number of access words; for I/O by BDI, the number of access word pairs.
buffer-addr	For standard I/O, the main storage address at which the transfer begins; for scatter/gather I/O, the address of the chain of access words; for I/O by BDI, the address of the chain of access word pairs.
G	Increment-decrement function designator as follows: 'D' – Decrementation 'N' – Inhibit incrementation and decrementation Omit this parameter for incrementation.
mass storage-addr	Identifies the logical mass storage address at which the operation starts.
search-sentinel	The sentinel to be recognized when a search function is performed on the mass storage file.

Example:

```
LRR          I$OD   'D222',BRD$ 56,BUFS,'D' 04333222
```

Filename D222 is entered in words 0 and 1 of the I/O packet. Function code 024, associated with the symbolic function code for a block read drum function (BRD$), is inserted in S2 of word 3. A total of 56 words are read into BUFS (address placed in H2 of word 4) with decrementation chosen (10_2 placed in G of word 4). The word transfer starts at drum address 4333222_8.

6.2.2.2. Mass Storage I/O Function with Interrupt

Purpose:

Create I/O packet for accessing mass storage files with interrupting functions.

Format:

```
I$OD    'filename',function ,interrupt-activity-addr,int-act-id word-count, buffer-addr [,G]
        mass-storage-addr [,search-sentinel]
```

Parameters:

Same as for I$OD (see 6.2.2.1) without interrupt, except that the interrupt addr and activity-id number are included in the parameters.

6.3. PROGRAM – I/O SYNCHRONIZATION

An activity is synchronized with the completion of an I/O operation, previously submitted by the same activity to the Executive through an IO$ request by entering the Executive through a WAIT$ request (see 6.3.1), a WANY$ request, or WALL$ request (see 6.3.2). The WAIT$ request may also be used to wait for the completion of an I/O operation performed by other program activities. When interrupt activities are used, they perform the I/O, not the originating activity.

A WAIT$ request waits for completion of a particular I/O operation and must be preceded by a Test Positive (TP) instruction on word 3 of the I/O packet (see Figure 6-1).

6.3.1. Wait for Completion of Specific I/O (WAIT$)

Purpose:

Delays execution of an activity until the I/O operation controlled by a specific I/O packet (see Figure 6-1) has been completed.

Format:

```
TP  pktaddr+3
ER  WAIT$
```

Description:

When an I/O Executive Request is submitted, the Executive sets word 3 of the I/O packet (see Figure 6-1) negative; word 3 remains negative until the completion of the I/O operation. The Test Positive (TP) check is made on this word.

Because the Executive performs a second test to determine the completion of the I/O request, the h- and i-designators of the Test Positive instruction must be set to zero.

The packet address is the specific request waited for at WAIT$.

6.3.2. Wait for Completion of any I/O (WANY$)

Purpose:

Delays execution of an activity until any I/O operation, issued by that activity has completed. No delay occurs if there are no outstanding I/O operations upon execution of the ER.

Format:

```
ER  WANY$
```

6.3.3. Wait for Completion of any I/O (WALL$)

Purpose:

Will cause a wait for the completion of all outstanding I/O operations for the program.

Format:

```
ER   WALL$
```

Description:

An ER to WALL$, is similar to an ER to WANY$ except control is not returned to the user until all I/O issued by the user's program has completed.

6.3.4. Initiate I/O and Return Control Immediately (IO$)

Purpose:

To request an operation on the file indicated and to return control to the executing program without waiting for completion of the I/O operation.

If the number of active I/O operations for the program becomes too large, the Executive turns the request into IOW$.

Format:

```
L,U  A0,pktaddr
ER   IO$
```

This linkage may be generated by the procedure call:

```
I$O pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6-1) which controls all I/O operations.

6.3.5. Initiate I/O and Return Control Immediately, with Interrupt (IOI$)

Purpose:

Same as for IO$ (see 6.3.4), except that an interrupt activity is initiated at completion of the I/O request.

If the number of active I/O operations for the program becomes too large, the Executive turns the request into IOWI$.

Format:

```
L,U   A0,pktaddr
ER    IOI$
```

This linkage may be generated by the procedure call:

```
I$OI  pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6-1) which controls all I/O operations.

6.3.6. Initiate I/O and Wait for Completion (IOW$)

Purpose:

Same as for IO$ (see 6.3.4), except that control is not returned to the requesting activity until completion of the I/O operation.

Format:

```
L,U   A0,pktaddr
ER    IOW$
```

This linkage may be generated by the procedure call:

```
I$OW   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6-1) which controls all I/O operations.

6.3.7. Initiate I/O and Wait for Completion, with Interrupt (IOWI$)

Purpose:

Same as for IOW$ (see 6.3.6), except that an interrupt activity is initiated upon completion of the I/O operation.

Format:

```
L,U   A0,pktaddr
ER    IOWI$
```

This linkage may be generated by the procedure call:

```
ISOWI   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6-1) which controls all I/O operations.

6.3.8. Initiate I/O and Exit, with Interrupt (IOXI$)

Purpose:

To request an operation on the file indicated and terminate the requesting activity. Upon completion, initiate an interrupt activity. This increases the completion priority and saves the time required to save and restore the register set.

Format:

```
L,U   A0,pktaddr
ER    IOXI$
```

This linkage may be generated by the procedure call:

```
ISOXI   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6-1) which controls all I/O operations.

6.3.9. Reducing Interrupt Activity Priority (UNLCK$)

Purpose:

Allows an interrupt activity to reduce its priority.

Format:

```
ER   UNLCK$
```

Description:

The UNLCK$ request enables an I/O interrupt activity to reduce its switching priority to the priority of the activity which initiated the I/O request. Any other Executive Request executed by the interrupt activity has the same result. However, in a time critical, multiactivity program, the UNLCK$ request provides a low overhead means of level reduction.

6.4. MAGNETIC TAPE HANDLER

6.4.1. Tape Handler Functions

The various magnetic tape functions are defined in Table 6–2. The current position of each tape is kept in terms of a block count and is made available for error logging, checkpoint, and ending label routines. No provision is made for automatic treatment of mixed parity and mixed density tape files.

Utilization of the skip write function is automatically provided by the handler (VIC/VIIIC only), and unless the user provides error recovery, these functions should not concern the user.

In order to use the handler, an I/O control packet must be generated (see 6.2).

Standard modes, in lieu of those set by the worker program, are established by the Executive at initialization and reestablished when a tape is released as follows:

- highest density possible
- odd parity
- no character translation
- 18–character noise constant
- standard recovery

In addition to the suppress recovery, the parity and density modes can be set by options on the @ASG or @MODE control statements (see 3.7.1 and 3.7.2).

Table 6-2. Magnetic Tape I/O Functions and Codes

Function	Symbol	Octal Code	Description
Write	W$	10	Starting at the address in H2 of word 4 of the I/O packet, transfer the number of words specified in H1 of word 4 to form a single block on magnetic tape. Transfer is accomplished according to the standard modes or the requested modes of parity, density, and so forth. Normal completion results when all words have been transferred.
Write by BDI	BDW$	04	Write a single block on magnetic tape specified by a string of access word pairs. The number of access word pairs is specified in H1 of word 4 of the I/O packet, and the starting address of the string is specified in H2 of word 4. The BDIs of the required based banks containing the data areas at which the transfers are to start is specified in H2 of the second word of each access word pair. See 6.4.2.6 for restrictions on scatter/gather.
Write end of file	WEF$	11	Write a sentinel on magnetic tape which, when read, results in an EOF status being returned to the program.
Skip write	SW$	13	Erase three inches of tape, then the same as a write function. This function is automatically provided in the system for write parity recovery. (Allowed on VIC/VIIIC only.)
Gather write	GW$	15	Write a single block on magnetic tape specified by a string of access words. The number of access words is specified in H1 of word 4 and the starting address of the string is specified in H2 of word 4. See 6.4.2.6 for restrictions on scatter/gather.
Read forward	R$	20	Initiate tape motion in the forward direction and transfer the words read into the area defined by word 4 of the packet. Transfer is normally concluded by either encountering the end of block or transferring the number of words requested.
Read backward	RB$	21	Same as read forward, except opposite direction.
Move forward	MF$	50	Move tape forward one block.
Move backward	MB$	51	Backspace the tape one block.
Forward Space File	FSF$	52	Move tape forward past the next EOF mark. It returns an EOF status if the end of the tape is not encountered.
Backspace File	BSF$	53	Move tape backward past the previous EOF mark. It returns an EOF status if the beginning of the tape is not encountered first.
Rewind	REW$	40	Reposition the tape to load point. This is the point at which a read forward reads the first block on tape and a read backwards reports an end-of-tape status.

Table 6-2. Magnetic Tape I/O Functions and Codes (continued)

Function	Symbol	Octal Code	Description
Rewind with interlock	REWI$	41	Reposition the tape to unload point and lock the unit against further functions.
Set mode	SM$	42	Set operating mode function (see 6.4.1).
Scatter read forward	SCR$	43	Same as read forward, except the words read are transferred into areas specified by a string of access words defined by word 4. See 6.4.2.6 for restrictions on scatter/gather.
Scatter read backward	SCRB$	44	Same as scatter read forward, except opposite motion direction. See 6.4.2.6 for restrictions on scatter/gather.
Read by BDI	BDR$	54	Same as read forward, except the words read are transferred into areas specified by a string of access word pairs defined by word 4. H2 of the second word of each access word pair gives the BDI for the required based bank containing the data area into which that transfer is to be made. See 6.4.2.6 for restrictions on scatter/gather.
Mode set	MS$	55	Set operating mode function (see 6.4.1).

6.4.1.1. Set Mode Function

For the set mode function (SM$), the I/O access control word (word 4 of Figure 6-1) is set to a 1-word buffer which defines the modes to be set as follows:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9
35	33	31	29	27	25	21	19	17 0

Field 1 - Density(s)

0_8 - No change
1_8 - 200 FPI (7-TRK tape); 800 FPI (9-TRK tape)
2_8 - 556 FPI (7-TRK tape); Illegal for 9-TRK tape.
3_8 - 800 FPI (7-TRK tape); 1600 FPI (9-TRK byte interface tape only)

Field 2 - Parity

0_8 - No change
1_8 - Odd (binary)
2_8 - Even (BCD)

Field 3 – Translate

0_8 – No change
1_8 – Set character translate mode
Fieldata to BCD for UNISERVO VIC/VIIIC
EBCDIC to BCD for byte interface
2_8 – Discontinue translation

Field 4 – Allow noise

0_8 – No change
1_8 – Set the noise constant to the number of characters in Field 9

Field 5 – Suppress recovery

0_8 – No change
1_8 – Return external interrupt status code to the worker program without attempting recovery
2_8 – Discontinue suppress recovery mode

Field 6 – MSA Translator

0_8 – No change
1_8 – Fieldata to/from EBCDIC
2_8 – Fieldata to/from ASCII
3_8 – XS-3 to/from EBCDIC
4_8 – XS-3 to/from ASCII
5_8 to 16_8 – Reserved for additional translate options
17_8 – Discontinue translation

Field 7 – Control unit data converter (byte interface only)

0_8 – No change
1_8 – Set data converter mode
2_8 – Discontinue data conversion

Field 8 – Byte channel transfer mode (see Table 6–3)

0_8 – No change
1_8 – Quarter-word (A format)
2_8 – 6-bit packed (B format)
3_8 – 8-bit packed (C format)

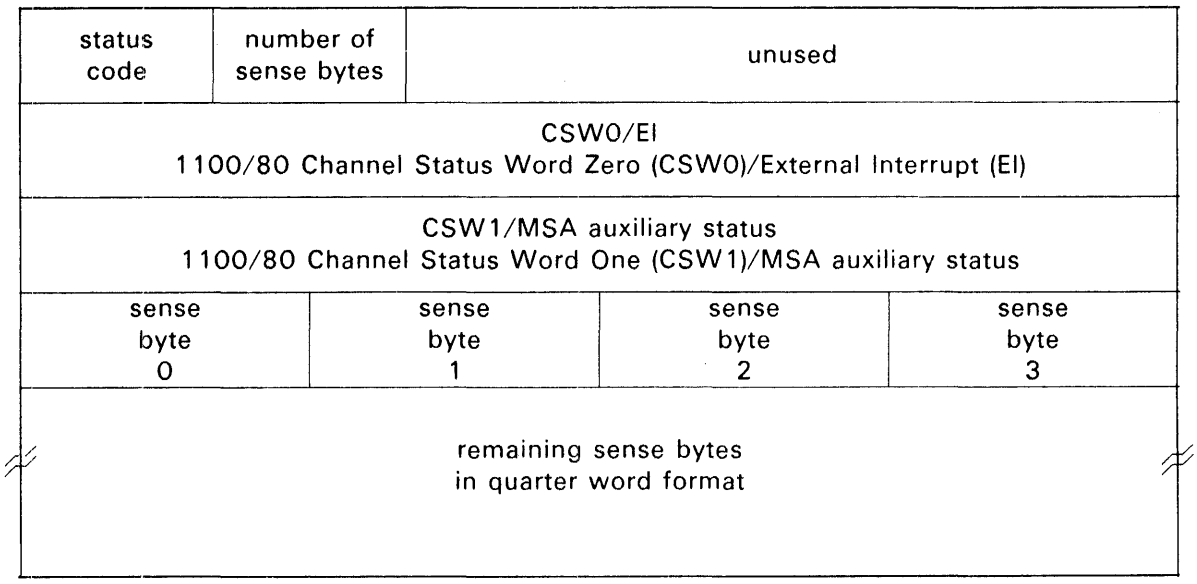
Field 9 – Noise constant character count

When suppress recovery mode is set, the following applies:

- Entry to I/O control is not possible via the IO$ interface.
- Only for the R$, RB$, SCR$, and SCRB$ functions is recovery suppressed.
- 011_8 is returned as the I/O packet status.
- For VIC/VIIIC tape units, register A1 of the waiting activity contains the EI status word.

For byte interface tapes only; if suppress recovery mode is being utilized, the tape I/O packet must be expanded by one word. This word contains a user-supplied access word which points to a status packet as shown below:

Suppress Recovery Status Packet



where:

status code

In order to reduce the device dependency of the suppress recovery user program, a partial check of the status is made by the Executive and a device independent status code is placed in the user's status packet. The status codes are as follows:

- 000 – Normal Completion (Channel End and Device End Only)
- 001 – End-of-File
- 002 – Data Check, Word Count Exceeds Noise Constant
- 003 – Data Check, Word Count Less than Noise Constant
- 004 – Other Error, Tape Moved
- 005 – Other Error, No Tape Motion
- 006 – Other Error, No Determination of Tape Motion by the Handler
- 007 – Unit Check, but Sense Bytes Could Not Be Obtained
- 010 – MSA Error, Have Auxiliary Status
- 011 – MSA Error, No Auxiliary Status
- 012 – Abnormal Byte Count, Have Auxiliary Status
- 013 – Abnormal Byte Count, No Auxiliary Status

6.4.1.2. Mode Set Function

The operation of the mode set function (MS$) is similar to that of the set mode function (SM$), except for translation and density change specifications. The I/O access control word (word 4 of Figure 6-1) points to a buffer of the following format:

00	MS$ flags
01	processor data format mnemonic
02	tape data format mnemonic

Word 0

The format of this word and the meanings of the MS$ flags are shown below:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	unused	Field 8
35	33	31	29	27	25	23 18	17 12	11 0

Field 1 - Control unit data converter (7-TRK byte interface only)

0 - No change
1 - Set data converter mode
2 - Discontinue data conversion

Field 2 - Parity

0 - No change
1 - Odd (binary)
2 - Even (BCD)

Field 3 - Translator

0 - No change
1 - Change translator to the processor/tape code specified in words 1 and 2 of the buffer pointed to by the user-supplied ACW.

Field 4 - Allow noise

0 - No change
1 - Set the noise constant to the number of characters in Field 8

Field 5 – Suppress recovery

0 – No change
1 – Return external interrupt status code to the worker program without attempting recovery
2 – Discontinue suppress recovery

Field 6 – Byte channel transfer mode (see Table 6-3)

0 – No change
1 – Quarter-word (A format)
2 – 6-bit packed (B format)
3 – 8-bit packed (C format)

Field 7 – Density

0 – No change
1 – 200 FPI (7-TRK tape); 800 FPI (9-TRK tape)
2 – 556 FPI (7-TRK tape); 1600 FPI (9-TRK byte interface tape only)
3 – 800 FPI (7-TRK tape); 6250 FPI (9-TRK byte interface tape only)

Field 8 – Noise constant character count. Noise constant value cannot exceed 100.

Word 1

The processor mnemonic defines the format of the data in the processor and is specified by one of the processor mnemonics listed below.

Word 2

The tape mnemonic defines the format of the data on the tape and is specified by one of the tape mnemonics listed below.

The following is the current list of processor/tape mnemonics that may be specified in words 1 and 2 of the buffer:

Processor Mnemonics	Tape Mnemonics
ASCII	BCD
ASCII	EBCDIC
ASCII	FLDATA
ASCII	XS-3
EBCDIC	BCD
FLDATA	ASCII
FLDATA	BCD
FLDATA	EBCDIC
FLDATA	XS-3
XS-3	ASCII
XS-3	BCD
XS-3	EBCDIC

The tape translator is turned off by specifying both processor/tape mnemonics to be the same.

If translation change is not specified, the buffer size should be reduced to one word.

6.4.2. General Considerations

The following must be considered when using the magnetic tape handler for compatible magnetic tape units.

6.4.2.1. Noise Constant Convention

In order to distinguish between "noise blocks" and actual data blocks, a value called the "noise constant" is defined. It is defined on the @ASG control statement (if not specified then a system default value is used) for magnetic tape units. It can be changed dynamically by use of the SM$ function, MS$ function, or the @MODE control statement.

The magnetic tape handler considers any block whose size is less than or equal to the noise constant as being a noise block. Furthermore, any requests for data transfers, where the total word count is less than or equal to the noise constant, is rejected with an 025_8 packet status.

To avoid the risk of data being read as noise, or vice versa, the noise constant should be set to the same value as when the tape was written.

6.4.2.2. Read Backward Limitations

The read backward function on the UNISERVO VIC/VIIIC or 7-track byte interface tape unit should not be used if the tape to be read has been recorded on some other type of unit. It is necessary that the recording produce a statically deskewed longitudinal check frame to prevent the read backward function from interpreting the check frame as data frames.

If a block is recorded in 7-track format with a block length greater than five frames and not a multiple of six, a read backward produces a different format than a read forward of the same block. For example: if the block length is seven frames, a read forward results in assembling frames 1 through 6 as the first word, and frame 7 as the second; and a read backward results in assembling frames 2 through 7 as the first word, and frame 1 as the second.

The same type of buffer variation exists for a read backward function on a 9-track unit if the write buffer length is not a multiple of two words (nine frames). A 1-word write on a 9-track unit results in five frames being recorded, with the fifth frame containing four bits of zero padding. A read backward results in the four bits of padding appearing as the least significant four bits of the first word assembled. Furthermore, regardless of the direction of reading, if a block is written on a 9-track format unit with an odd word count in the access word, one more word is made available as input than was sent out to be written.

6.4.2.3. Write Considerations

Care must be exercised to avoid unwanted truncations of write transfers by means of a "stop code". For word interface tape units with even parity set, a stop code is a data frame of 00_8 if BCD translate is specified; otherwise, it is 46_8. For byte interface tape units, a stop code is the ninth bit of the character field when using A format.

The unwanted truncations can result in an unrecoverable late acknowledgement condition or in writing a noise block. The latter is not detected until the tape is read; in which case, erroneous data is received.

To avoid unwanted truncations of write transfers, the use of the "stop code" should be confined to the last word of data to be written.

When writing in even parity mode on word interface tape units, the following should be considered:

- If an attempt is made to write an EOF or a zero-length block by means of the "stop code", the request is rejected with a 25_8 status in the IO packet.
- When software BCD translate is specified, the original contents of the data buffer is destroyed.
- Even-parity writes on compatible tape units using the "stop code" to terminate data transfer results in an erroneous final-word-count-returned-by-I/O in the I/O packet (Figure 6-1) whenever the stop code is contained in other than the last word to be transferred from the output buffer. If the stop code is contained in the first or sixth character position, the final-word-count is four words greater than the actual number of words transferred. If the stop code is contained in any of the second through fifth character positions, the final-word-count is three words greater. The final-word-count does not exceed that contained in the output access word.

When using the byte interface tape units, there are a number of incompatibilities with the UNISERVO VIC/VIIIC tape operations of which the programmer should be aware. They are:

- Fieldata-to-BCD translations do not convert all the codes the same.
- The Fieldata 00_8 code in even parity does not stop the write operation as on the C-type units. Instead it is converted to a 20_8 (or 14_8 if BCD translator is on).
- A variable length block by character can be accomplished by using the A format and setting the ninth bit of the character field. This stops the write operation and does not record that character containing the stop bit. It is illegal to set the first stop bit since this is equal to an I/O of zero words. On any format it is required that the access word terminate on the word containing the first stop bit if the stop bit is being used.
- When using the C format read and a block ends on a fifth or sixth frame, the even word is not transferred to the processor (see Table 6-3).
- A status of 05 is returned to the user if an error is detected in translation.

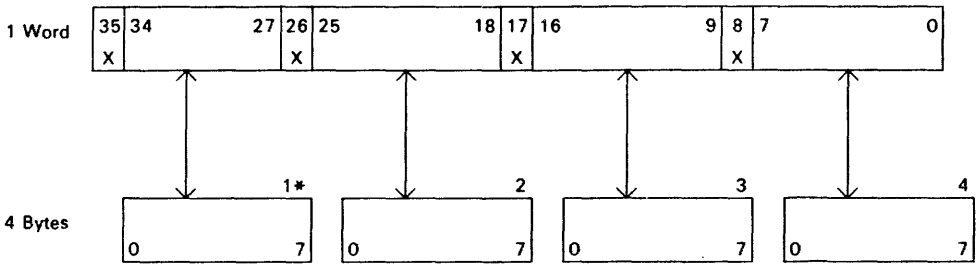
The recovery procedure for a parity error or certain tape hash errors on a write operation may utilize four feet of tape or twice the length of the block, whichever is larger. Hence, if blocks are to be recorded which are longer than four feet (or less, depending upon whether an ending interrupt activity submits the next request or if requests are queued ahead by I/O control), it is recommended that tapes be used which have the end-of-tape mark at least 10 feet from the actual end of the tape to ensure that the tape is not pulled off the supply reel.

6.4.2.4. Move Considerations

The MF$, MB$, FSF$, and BSF$ functions are concerned with position. Parity errors are not reported and are only examined when necessary to determine if a block is an EOF block.

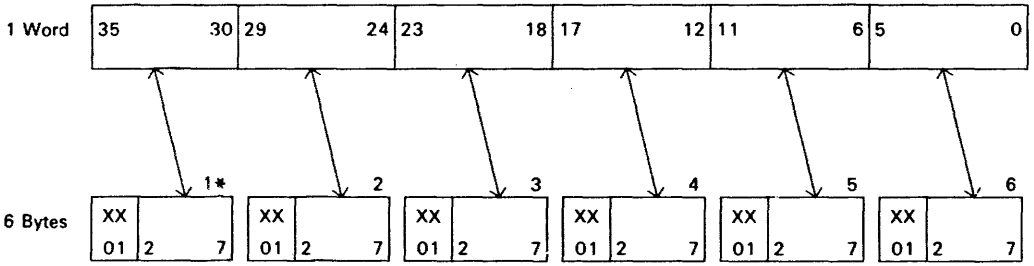
Table 6-3. Byte Channel Data Word Formats

FORMAT A (QUARTER WORD)



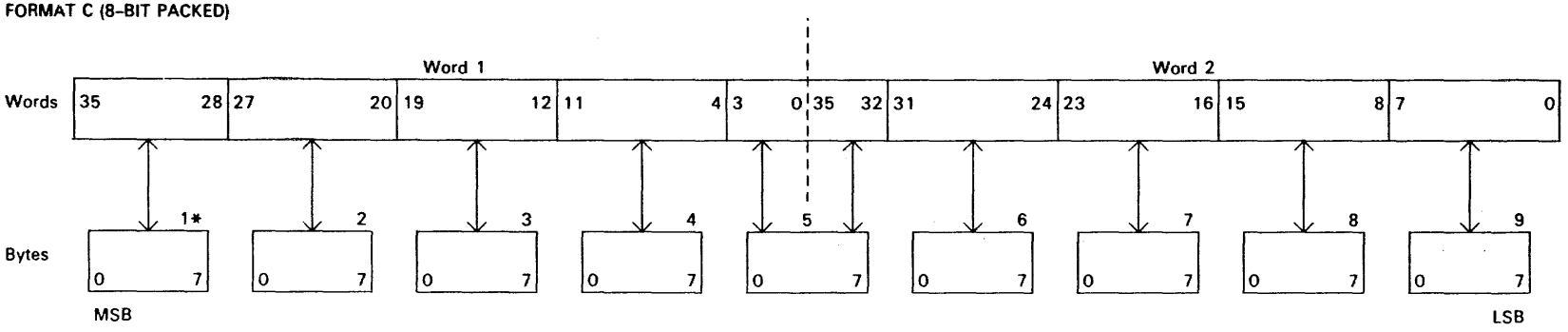
NOTE: Bits 35,26,17, and 8 are used for stop control on output operations and forced to binary 0 on input operations.

FORMAT B (6-BIT PACKED)



NOTE: Bit 0 and 1 become binary 0 on output and are ignored on input, for each 8-bit byte. When translation is specified, bits 0 and 1 are not forced to binary 0.

FORMAT C (8-BIT PACKED)



*Numbers on arrows indicate the order of byte transfer.

6.4.2.5. Abnormal Frame Count Considerations

The AFC (abnormal frame count) field in the I/O packet (see word 3 of Figure 6-1) is supplied by the Executive when a status code of 4 is returned. The value of this field is determined as follows:

- If the character count is not a multiple of 6 on 7-track tapes, AFC contains the number of characters in the last word read from the tape.
- For 9-track tapes, the count is the number of 8-bit bytes assembled and transferred to the CPU in the last 2-word sequence.
- Examples:

  1. Given a 101-word block on a 9-track tape. Reading it (on either a 9-track UNISERVO VIC or VIIIC or UNISERVO 12/14/16/20/30/32/34/36 in C format) results in the following:

     a. ACW word count of 101, gives a 0 status, AFC = 0 and a word count of 101.

     b. ACW word count greater than 101 gives a 4 status, AFC = 5 and a word count of 102.

  2. Writing a 100-word block on a 9-track tape and reading it results in a 0 status regardless of the word count.

6.4.2.6. Scatter-Read/Gather-Write Considerations for High Speed Tapes

Scatter-read and gather-write operations are not supported on 1106, 1108, 1100/10/20 Systems for the UNISERVO 20 tape units due to the high transfer rate of the device (320,000 bytes/sec.) If a scatter-read (SCR$), scatter-read backward (SCRB$), gather-write (GW$), write by BDI (BDW$), or read by BDI (BDR$) function is detected and the tape to be referenced is a UNISERVO 20 tape unit, the requestor is taken to the error mode routine with an error type 1 and code of 024. Scatter/gather operations for the UNISERVO 20 tape unit are supported on 1110, 1100/40 Systems.

Because of hardware restrictions on UNIVAC 1100/10, 1100/20, and 1106 systems operating in extended addressing mode (524K), all I/O read and write operations with word counts greater than 32,767 words are split up into scatter-read or gather-write operations. Thus, a read or write operation with a word count greater than 32,767 to a UNISERVO 20 tape unit is not supported for these systems.

Several restrictions have been made in the area of scatter/gather support on all UNISERVO 30 and block lengths on UNISERVO 36 tape units.

- The UNISERVO 36 is not supported with an unbuffered control unit.

- The UNISERVO 36 is supported with the buffered control unit but limits have been placed on the block size. The following chart shows the block lengths supported on each Series 1100 System with each buffer size:

Restrictions – UNISERVO 36 Tape Unit Block Length

System Type (5042 buffer size)	1106-I, 1110*	1108, 1106-II, 1100/10/20, 1100/40**
Unbuffered	Not Supported	Not Supported
1K Buffer	Not Supported	4200 Words
2K Buffer	4200 Words	8400 Words

* Assumes storage types 7005B, 7013, or storage units which have a cycle time greater than 1.2 microseconds.

** Assumes storage types 7005, 7005A, 7015, 7033A, 7033B, 7030A, 7030B, 7036A, 7036B, or storage units which have a cycle time equal or less than 1.2 microseconds.

No block length restrictions on 5042 control units are connected to 1100/80 Systems.

- Scatter/Gather is supported on all systems when a buffered 5042 control unit is utilized.
- Scatter/Gather is not supported on the UNISERVO 30 at 1600 bpi nor on the UNISERVO 32/UNISERVO 34 at 6250 bpi when an unbuffered 5042 is utilized and the system does not have hardware data chaining. The following chart illustrates this restriction:

Restrictions – Scatter/Gather Support (Unbuffered Control Unit)

Servo / System	U30 1600 BPI	U32 1600 BPI	U32 6250 BPI	U34 1600 BPI	U34 6250 BPI
1106/08/ 10/20	No	Yes	No	Yes	No
1100/40/80 1110	Yes	Yes	Yes	Yes	Yes

There are no restrictions for 1100/80 Systems.

See Table 6–4 for magnetic tape functions versus unit types.

Table 6-4. Magnetic Tape Function versus Unit Type

Function	Symbol/Octal Code	Packet Size	VIC VIIIC	7-Track 12/14/16	9-Track 12/14/16/20/30	9-Track 32/34/36
Read Forward	R$/20	5	(1)	(1)	(1)	(1)
Read Backward	RB$/21	5	(1)	(1)	(1)	(1)
Read by BDI	BDR$/54	5	(1)	(1)	(3)	(3)
Scatter Read	SCR$/43	5	(1)	(1)	(3)	(3)
Scatter Read Backward	SCRB$/44	5	(1)	(1)	(3)	(3)
Move Forward	MF$/50	4	(1)	(1)	(1)	(1)
Move Backward	MB$/51	4	(1)	(1)	(1)	(1)
Write	W$/10	5	(1)	(1)	(1)	(1)
Write by BDI	BDW$/04	5	(1)	(1)	(3)	(3)
Write end-of-file	WEF$/11	4	(1)	(1)	(1)	(1)
Skip Write	SW$/13	5	(1)	(2)	(2)	(2)
Gather Write	GW$/15	5	(1)	(1)	(3)	(3)
Rewind	REW$/40	4	(1)	(1)	(1)	(1)
Rewind with Interlock	REWI$/41	4	(1)	(1)	(1)	(1)
Forward Space File	FSF$/52	4	(1)	(1)	(1)	(1)
Backspace File	BSF$/53	4	(1)	(1)	(1)	(1)

Table 6-4. Magnetic Tape Function versus Unit Type (continued)

Function	Symbol/Octal Code	Packet Size	VIC VIIIC	7-Track 12/14/16	9-Track 12/14/16/20/30	9-Track 32/34/36
Set Mode/Mode Set:	(SM$/MS$)42/55	5/6	(1)	(1)	(1)	(1)
6250 FPI			(2)	(2)	(2)	(1)
1600 FPI			(2)	(2)	(1)	(1)
800 FPI			(1)	(1)	(1)	(2)
556 FPI			(4)	(1)	(2)	(2)
200 FPI			(4)	(1)	(2)	(2)
Odd parity			(1)	(1)	(1)	(1)
Even parity			(4)	(1)	(2)	(2)
Translate			(4)	(1)	(1)	(1)
Allow noise			(1)	(1)	(1)	(1)
Suppress recovery			(1)	(1)	(1)	(1)

CODE: (1) Available.

(2) Invalid function, causes termination.

(3) Not available for high speed tapes with unbuffered units and slow processor storage (see 6.4.2.6).

(4) Applicable to 7-track tapes VIC/VIIIC only.

6.5. WORD-ADDRESSABLE FORMAT MASS STORAGE

6.5.1. Word-addressable Format Functions

Two general modes of mass storage device operation are provided within the executive. The first is sector-formatted mass storage which allows execution of a program with files designed for sector-formatted mass storage to. The second mode is as a word random storage device word addressable. The interpretation of function codes for sector-formatted mass storage is discussed in subsection 6.6. For word addressable format, the functions are listed in Table 6-5.

6.5.2. General Considerations

The functions listed in Table 6-5 are performed on areas reserved through the use of the @ASG control statement (see 3.7.1).

The executive ensures that a search find is within the assigned area before reading thus guaranteeing file privacy. If a read after search must be truncated, a status code of 1 is returned to the program. A search function issued from a user's program searches a maximum of one granule of the file. If a find is made, it must be within the same granule in which the search was started. A search read drum or block search read drum is terminated if the access control word is zero. When the read phase of the SRD$ or BSRD$ functions involves going over granule boundaries, the read is completed for the user. If any part of a read after search find is outside of the assigned area, the request is truncated.

The I/O status code 5 for word addressable I/O requests results from attempting I/O in an area of a file not currently allocated. A status code of 22_8 does not occur unless the request starts beyond the highest word written.

Table 6-5. Word-Addressable Mass Storage I/O Functions and Codes

Function	Symbol	Octal Code	Description
Write	W$	10	Starting at the main storage address specified in H2 of word 4, transfer the number of words specified in H1 of word 4 to the mass storage area starting at the relative word address in word 5 of the I/O packet. The write operation also removes any locks on the area written in the same manner as the unlock operation. The write operation expands the file automatically up to the maximum on the @ASG control statement.
Write by BDI	BDW$	04	Transfer the number of words specified by a string of access word pairs, specified by word 4 of the I/O packet, to the mass storage area starting at the relative word address in word 5. The required based bank containing the data area to be transferred is given by the BDI in H2 of the second word of each access word pair. The number of access word pairs is specified in H1 of word 4, and the address of the access word pairs is specified in H2 of word 4. Any locks on the area written are removed as on a GW$.
Gather write	GW$	15	Transfer the number of words specified by a string of access words specified by word 4 from the areas specified by these access words to the drum area starting at the relative word address in word 5. The number of access words is specified in H1 of word 4, and the address of the access words is specified in H2 of word 4. The write operation also removes any locks on the area written in the same manner as the unlock operation.
Acquire	ACQ$	16	Starting at the relative address specified in word 5 of the I/O packet, the file is expanded by the number of granules required to hold the number of words specified in H1 of word 4 if the granules are not already allocated. This allows expansion of a file without writing into it.
Extended Acquire	EACQ$	17	Starting at the file relative address specified in word 5 of the I/O packet, the file is expanded by the number of tracks (1 track = 1792 words) specified in the word-count field of word 4. Optionally, the function may cause the allocation to be made on the device or pack specified in word 7 and device or pack relative address specified in word 8. The versatility of the function is controlled by entry flags specified in word 6,S3. In addition, the status bit(s) specified in word 6,H2 will be set in the internal entry created to describe the allocation. (See 6.5.3 for a detailed description of the function.)
Read	R$	20	Starting at the relative word address in word 5 of the request packet, transfer the number of words in H1 of word 4 into the area starting at the address in H2 of word 4. Normal completion (status 0_8) indicates the specified number of words have been transferred to main storage from word 4.
Scatter read	SCR$	43	Starting at the relative word address in word 5 of the I/O packet, transfer the number of words specified by a string of access words defined by word 4 to the areas specified by these access words. The

Table 6-5. Word-Addressable Mass Storage I/O Functions and Codes (continued)

Function	Symbol	Octal Code	Description
			number of access words is specified in H1 of word 4, and the address of the access words is specified in H2 of word 4.
Read and Release	RR$	22	Same as read with the additional condition that, after the read has been performed, all granules with any part within the set of addresses described by the packet are released to the available mass storage pool.
Release	REL$	23	Same as read and release, except no reading is performed.
NOTE: *The following functions, BRD$, BSRD$, SD$, SRD$, and BSD$, are included for compatibility only and should not be used in new programs.*			
Block read	BRD$	24	Starting at the relative word address in word 5 of the I/O packet, transfer words from a word-addressable drum file to main storage at the address in H2 of word 4 until either the number of words specified in H1 of word 4 has been read, or until the end-of-block sentinel (a word of all 1's) is read. Encountering a sentinel is noted by 1_8 status code, and the sentinel word is transferred as the last word in the buffer. The substatus field (H2 of WORD 3) indicates the number of words read. If completion is due to end-of-block and the buffer length is such that another word can be accepted, the overflow word (the word on the device following the sentinel) is stored in the buffer following the sentinel word. The upper six bits of the overflow word, if stored, are set to 04. Termination of the read may also be caused by end-of-file or end-of-assignment (both conditions returning on 02 status), or an abnormal I/O condition.
Read and lock	RDL$	25	Perform the read operation and impose a logical lock on the area read, which prevents access to the part of the file defined by the access word and relative starting address by other activities (of either the same run or other runs) until such time as the locking activity unlocks the area. Removal of this exclusive use of a block is accomplished by writing into any part of the block, issuing an unlock request or by terminating the activity (see 6.6).
Unlock	UNL$	26	Remove any logical locks imposed on other activities by read and lock requests submitted by this activity for the area of the file specified by the address and length of the access for this request. Locks are maintained by block, and unlocking any part of a block unlocks the entire block. Also, one unlock request can unlock several blocks (see 6.6).
Block search read	BSRD$	37	Starting at the relative word address in word 5 of the I/O packet, compare equal between the words in a word-addressable file and word 6 of the I/O packet. If a match is found before end-of-block sentinel, end-of-file, end-of-assignment, or end-of-granule (track or position), the relative address of the found word is stored in word 7 of the packet, and a status of 0 is returned. Words are transferred

Table 6-5. Word-Addressable Mass Storage I/O Functions and Codes (continued)

Function	Symbol	Octal Code	Description
			to the user as in block read, with truncation for end-of-block sentinel, end-of-file, end-of-assignment, or end-of-granule. Storing the overflow word and the status word, follows the same criteria as for the block read. If a match is not found, a status of 03 is returned and no transfer takes place.
Search	SD$	34	Starting at the relative word address in word 5 of the I/O packet, compare all words on a word-addressable file until a compare equal is made with word 6 of the packet, or until end-of-file, end-of-assignment, or end-of-granule (track or position) is reached. If a find is made, the relative address of the found word is stored in word 7 of the packet and a status of 0 is returned. A status of 3 is returned if a find is not made.
Search read	SRD$	36	Starting at the relative word address in word 5 of the I/O packet, compare all words on a word-addressable file until either a compare equal is made with word 6 of the packet or until end-of-file, end-of-assignment, or end-of-granule (track or position) is reached. If a find is made, store the relative address of the found word in word 7 of the packet, and return a status of 0. Words are transferred to the user as in block read, with the exception that an end-of-block sentinel is not a termination condition, nor is status returned for it. A status of 03 is returned if a find is not made, and no transfer takes place.
Block search	BSD$	35	Same as the search read, with the exception that the end-of-block sentinel is taken as a termination condition, and that no transfer to the user takes place. The 03 status is also returned if the find is not made before the end-of-block.
Read by BDI	BDR$	54	Starting at the relative word address in word 5 of the I/O packet, transfer the number of words specified by a string of access word pairs, defined by word 4, to the areas specified by these access word pairs. The required based bank containing the data area to be transferred to is given by the BDI in H2 of the second word of each access word pair. The number of access word pairs is specified in H1 of word 4, and the address of the access word pairs is specified in H2 of word 4.

6.5.3. EACQ$ Function

The following description of the I/O packet for the extended acquire function (EACQ$) applies to all mass storage files. Words 0 through 5 of the packet are as defined in 6.1.1 with any exceptions noted herein. The function requires that the caller be privileged; i.e., the caller must have the file SYS$*DLOC$ assigned with the proper keys.

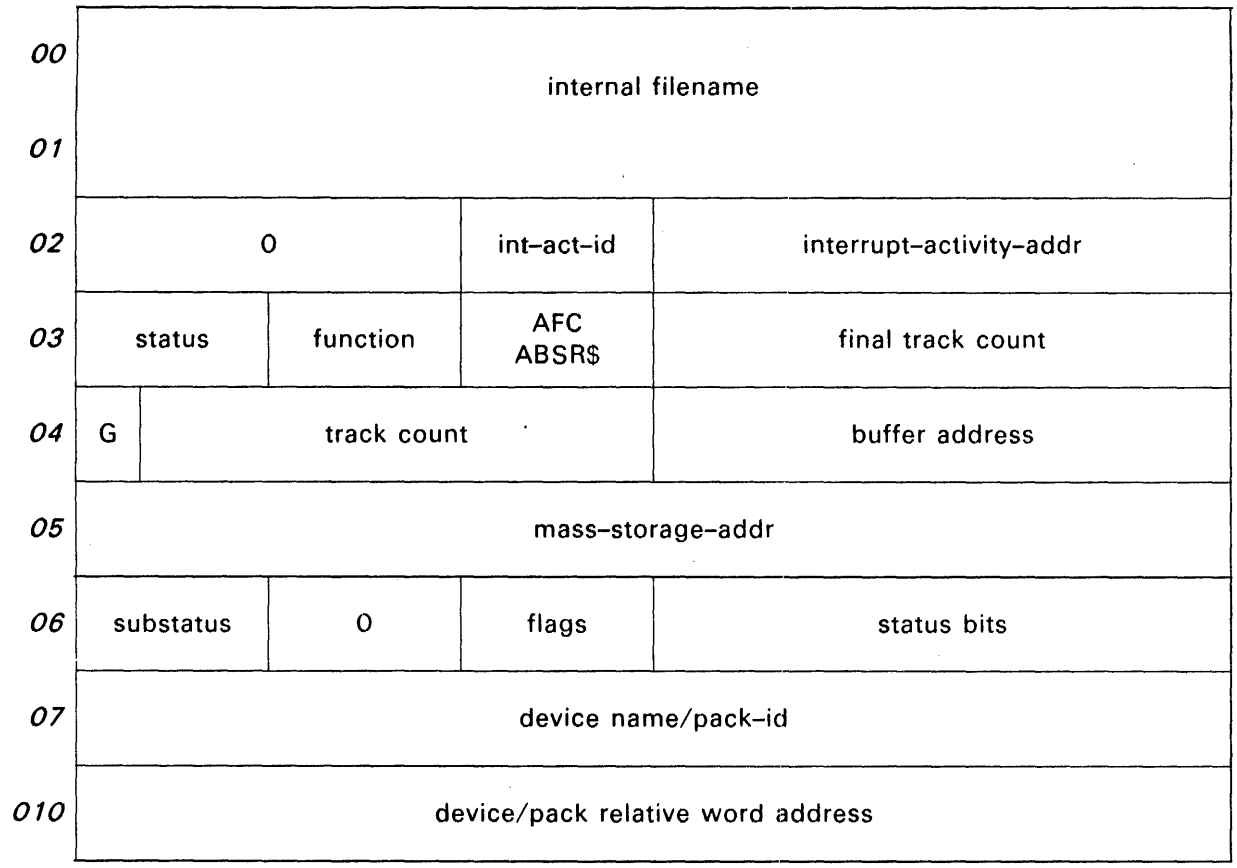


Figure 6-2. I/O Packet for EACQ$ Function

Word 04—H1

Track count — Number of tracks of mass storage requested with one track representing 1792_{10} words. The most significant two bits (which define the G field for an ACW) are ignored. Note that the file must have track granularity to permit it to expand in tracks rather than positions.

Word 05

Mass storage address — File relative address where allocation is to start with the address in sectors for a sector-formatted file, and in words for a word addressable file. This address must be on a track boundary.

Word 06

Substatus — This is valid only when a status of 2 is returned in word 3. The substatus provides appropriate clarification for rejection of a request and has the following definitions:

01 Mass storage address not an integral of tracks.
02 Device/pack relative address not an integral of tracks.
03 Nonexistent or invalid device name/pack-id specified.
04 Device name specified for disk mass storage.

010 Invalid flags contained in word 6 as defined in the following:

1. Device name and pack-id flags both set.
2. Device relative address flag set without having the device name or pack-id flag set.
3. Mandatory request flag set without having the device name or pack-id flag set.

011 The device or pack is not available for allocation.
012 Device or mass storage overflow.
013 A portion of the file which the request is attempting to allocate has already been allocated.
014 The range of device/pack relative addresses encompassed by the request do not exist.
015 The device/pack relative address specified is not available for allocation.
016 Contiguous mass storage is not available for allocation.

Flags

This contains binary coded parameters with the following definitions:

01 Mandatory Request Flag. This flag indicates that the request must be performed exactly as specified and as a single contiguous allocation or it must be rejected with an appropriate status. Setting this flag requires that a device name or pack-id be specified. If this flag is not set, the allocation need not be contiguous or at any specific device relative address should one be specified.
02 Device Name Flag. This flag indicates a device name has been specified in word 7 of the packet.
04 Device/Pack Relative Address Flag. This flag indicates a device/pack relative address has been specified in word 8 of the packet.
010 Pack-id Flag. This flag indicates a pack-id has been specified in word 7 of the packet.

Status Bits

This specifies the status bits to be inserted in the entry created to describe this allocation. Their definitions are:

010 DNR (Do Not Read)
020 DNW (Do Not Write)

The DNR and DNW bits are used to prevent the reading or writing, respectively, of specific file relative areas described by individual DAD entries.

Word 07

Device Name/Pack-id

This is optional and is used to specify a Fieldata device name or pack-id as indicated by the device name and pack-id flags. Only a pack-id may be specified for disk mass storage. An attempt to specify a device name for disk will result in the request being rejected for invalid parameter flags. The request will be rejected with an appropriate status if it is not possible to perform the allocation on the device or pack specified.

Word 010

Device/Pack Relative Word Address	This is optional as indicated by the device relative address flag. It is the relative word address at which to start the allocation and it must be an integral number of tracks (on a track boundary). If this word is supplied, a device name or pack-id must also be specified. If the allocation cannot be performed at this address and the mandatory request flag is not set, the allocation will be made in any available area on the device or pack specified.

6.6. SECTOR-FORMATTED MASS STORAGE

The various sector-formatted mass storage I/O functions are listed in Table 6-6.

Space on sector-formatted mass storage is assigned in granules of one track (64 sectors) or one position (64 tracks). A file consisting of more than one granule may be considered contiguous by the programmer because the I/O complex takes care of the processing that must occur whenever a granule boundary is passed. The I/O complex works in conjunction with the file supervisor to convert the relative sector addresses supplied by the user program into physical channel, unit, position, and sector addresses.

Table 6-6. Mass Storage I/O Functions and Codes

Function	Symbol	Octal Code	Description
Write	W$	10	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified in H1 of word 4 from the main storage area starting at the address in H2 of word 4 to sector-formatted mass storage. If the count is not a multiple of 28, write zeros into the remainder of the last sector. If the area being written into is not currently assigned, expansion of the file is automatic up to the maximum from the @ASG control statement (see 3.7.1). The write operation also removes any locks on the area written in the same manner as the unlock operation.
Write by BDI	BDW$	04	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified by a string of access word pairs, defined by word 4, to sector-formatted mass storage. The required based bank containing the data area to be transferred is given by the BDI in H2 of the second word of each access word pair. The number of access word pairs is specified in H1 of word 4, and the address of the access word pairs is specified in H2 of word 4. If the word count is not a multiple of 28, write zeros into the remainder of the last sector. If the area being written into is not currently assigned, expansion of the file is automatic up to the maximum from the @ASG control statement (see 3.7.1.). The write operation also removes any locks on the area written in the same manner as the unlock operation.
Gather write	GW$	15	Same as write, except word 4 specifies a string of access words, each specifying a word count and a main storage area.

Table 6-6. Mass Storage I/O Functions and Codes (continued)

Function	Symbol	Octal Code	Description
Acquire	ACQ$	16	Starting at the relative sector address specified in word 5 of the I/O packet, the file is expanded by the number of granules required to hold the number of words specified in H1 of word 4 if the granules are not already allocated. This allows expansion of a file without writing into it.
Extended Acquire	EACQ$	17	Starting at the file relative address specified in word 5 of the I/O packet, the file is expanded by the number of tracks (1 track = 1792 words) specified in the word-count field of word 4. Optionally, the function may cause the allocation to be made on the device or pack specified in word 7, and the device or pack relative address specified in word 8. The versatility of the function is controlled by entry flags specified in word 6,S3. In addition, the status bit(s) specified in word 6,H2 will be set in the internal entry created to describe the allocation. (See 6.5.3 for a detailed description of the function.)
Read	R$	20	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified in H1 of word 4 into the main storage area starting at the address in H2 of word 4. Reading always starts at a sector boundary but may end anywhere.
Read by BDI	BDR$	54	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified by a string of access word pairs, specified by word 4, into the main storage areas specified by the access word pairs. The required based bank containing the data area to be transferred to is given by the BDI in H2 of the second word of each access word pair. The number of access word pairs is specified in H1 of word 4, and the address of the access words is specified in H2 of word 4.
Scatter read	SCR$	43	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified by a string of access words specified by word 4 into the main storage areas specified by the access words. The number of access words is specified in H1 of word 4, and the address of the access words is specified in H2 of word 4.
Read and release	RR$	22	Same as read with the additional condition that, after the read has been performed, all granules with any part within the set of addresses described by the packet are released to the available mass storage pool.
Release	REL$	23	Same as read and release, except no reading is performed.
Read and lock	RDL$	25	Perform the read operation and impose a logical lock to be placed on the area read, which prevents access to the part of the file defined by the access word and relative starting address by other activities (of either the same run or other runs) until such time as the locking activity unlocks the area. Removal of this exclusive use of a block

Table 6-6. Mass Storage I/O Functions and Codes (continued)

Function	Symbol	Octal Code	Description
			is accomplished by writing into any part of the block, issuing an unlock request, or terminating the activity.
Unlock	UNL$	26	Remove any logical locks imposed on other activities by read and lock requests submitted by this activity for the area of the file specified by the address and length of the packet for this request. Locks are maintained by block, and unlocking any part of a block unlocks the entire block. Also, one unlock request can unlock several blocks.
NOTE: *The following functions, TSA$, TSF$, PSA$, and PSF$, are included for compatibility only and should not be used in new programs.*			
Track search all words	TSA$	30	Starting at the relative sector address in word 5 of the I/O packet, compare each word in a sector-formatted mass storage file with the identifier in word 6 of the packet until either a compare equal is made or the end-of-file, end-of-assignment, or end-of-track (sector address which is the next multiple of 100_8) is encountered. If a compare equal is found, store the relative sector address of the sector in which the find is made in word 7 of the packet, return a status of 0, and read as many words as are specified in H1 of word 4 (or to end-of-file, or end-of-assignment, whichever is smaller) starting with the beginning of the sector in which the find was made. If no compare equal is made before end-of-track, end-of-file, or end-of-assignment, a status of 03 is returned and no transfer takes place.
Track search first word	TSF$	31	Same as track search all words, except a comparison is made only on the first word of each sector.
Position search all words	PSA$	32	Same as track search all words, except comparisons are made until a sector address which is a multiple of 10000_8, is reached.
Position search first word	PSF$	33	Same as position search all words, except comparisons are made only on the first word of eacn sector.

An attempt to read from an area of a file which is not entirely assigned results in a status code of 5 being returned to the I/O packet. If the area starts within the assignment and runs beyond it, the substatus count H2 of Word 3 reflects the part assigned. If granules have been released causing voids within the file, a request could generate a legal start and ending address but a void within the file and this would result in the 5 status being returned with only the first part of the file read. Writing into an unassigned area of a sector-formatted file causes space to be assigned to the portion of the file. The automatic expansion on a write function continues until limited hy the maximum granule parameter on the @ASG control statement (see 3.7.1). In this case, a status code of 22_8 is returned in the I/O packet.

Portions of sector-formatted files can be exclusively assigned temporarily to a run by using the read and lock mechanism (see 6.5.1). Thus an activity can read an area of the file and update it without

having another activity of any program, to which the file is assigned, access the volatile data. The exclusive-use feature applies only to the area being accessed. All other areas of the file can be accessed during this period of exclusive use. The only activity permitted to access the locked out area is the activity that initially set the exclusive-use option. This activity may read, write, or unlock the locked out area.

Since the exclusive use of files by block (as defined by the address and access word) involves an interaction between activities, the user should ensure that proper order is maintained in submitting requests to prevent two activities from locking against each other. To aid in detecting this interlock condition, I/O control checks the length of time that an activity leaves a lock on an item. If an item is locked by any one activity for over 12 minutes, at the time of the unlock sequence (either a write or unlock function) a status code (10_8) is returned to the I/O packet, indicating that exclusive use had timed out and has been removed. Removing exclusive use by this means allows the locked activities to progress in the normal manner and the locking activity no longer interferes. If the unlock operation is the result of a write request, the write function is not performed and the 10_8 status code (see 6.9) is returned. The 12-minute limit for locking a file area can be changed at a particular site via a configuration parameter. A minute in this context is defined as 10 dayclock interrupts. On the 1110, 1100/10/20/40/80 Systems, 10 dayclock interrupts are equivalent to 65.536 seconds.

During normal operation, the handler prepositions the various units to keep access time to a minimum. For this reason, the position function is not needed in the user's repertoire.

6.7. DISK MASS STORAGE

All functions that are supported for word-addressable (see Table 6-5) and sector-formatted mass storage are supported on disk (see Table 6-6). That is, disk fully supports both sector-formatted and word-addressable files. The disk hardware provides for direct support of standard read and write operations, but does not provide for block type operations (e.g., BSRD$) and search operations. Thus, the following functions are simulated by software:

1. Block Read (BRD$)
2. Block Search (BSD$)
3. Block Search Read (BSRD$)
4. Search (SD$)
5. Track Search First Word (TSF$)
6. Track Search All Words (TSA$)
7. Position Search First Word (PSF$)
8. Position Search All Words (PSA$)

NOTE:

Block type operations and search operations should not be used in new programs.

The above functions are not recommended for disk use.

Space is allocated in granules of one track (64 sectors), or one position (64 tracks) as denoted on the @ASG statement (see 3.7.1). The file space may be considered contiguous by the programmer in that the I/O complex handles any discontinuities that exist.

The system provides for the mounting and dismounting of disk packs (other than those specified as part of the fixed mass storage pools). This feature benefits the installation in that it can have an infinite amount of mass storage space. It benefits the user program in that files can be assigned to specific disk packs. The Executive provides protection in this area so as to secure packs from unauthorized access.

Because of the ability to specify file placement, the programmer should be aware of the addressing mechanism provided on the packs. Disk packs are prepped, normally, so as to allow four sectors/record with 12 records/track (8414, 8424, 8425); 22 records/track (8440); 20 records/track (8430, 8433, 8434); 16 records/track (8405); or 29 records/track (8450). The system provides the ability to, if desired by the individual user, prep the program's packs in either of three other methods (i.e., one sector/record, two sectors/record, or sixteen sectors/record). The alternate prep factors are provided if the programmer finds that the standard prep factor is not optimal for application. Note, however, that alternate prep factors of two or less sectors/record cause a considerable loss in usable data space and some loss in effective transfer rate. This should be taken into account when considering an alternate prep factor.

Once a pack is prepped, there should never be a need to reprep the pack. However, the Executive does provide an override so as to enable reprepping, if necessary. The prepping of a pack is specified only at the system console and not internal to the program. Thus, the program need not concern itself with prepping in that it has already been done.

The user program need not concern itself with accessing part of a multisector record. However, write operations not on record boundaries require one or two internal reads which lengthen the time needed to perform an I/O request.

6.8. ARBITRARY DEVICE INTERFACE (ADI) (NON-1100/80)

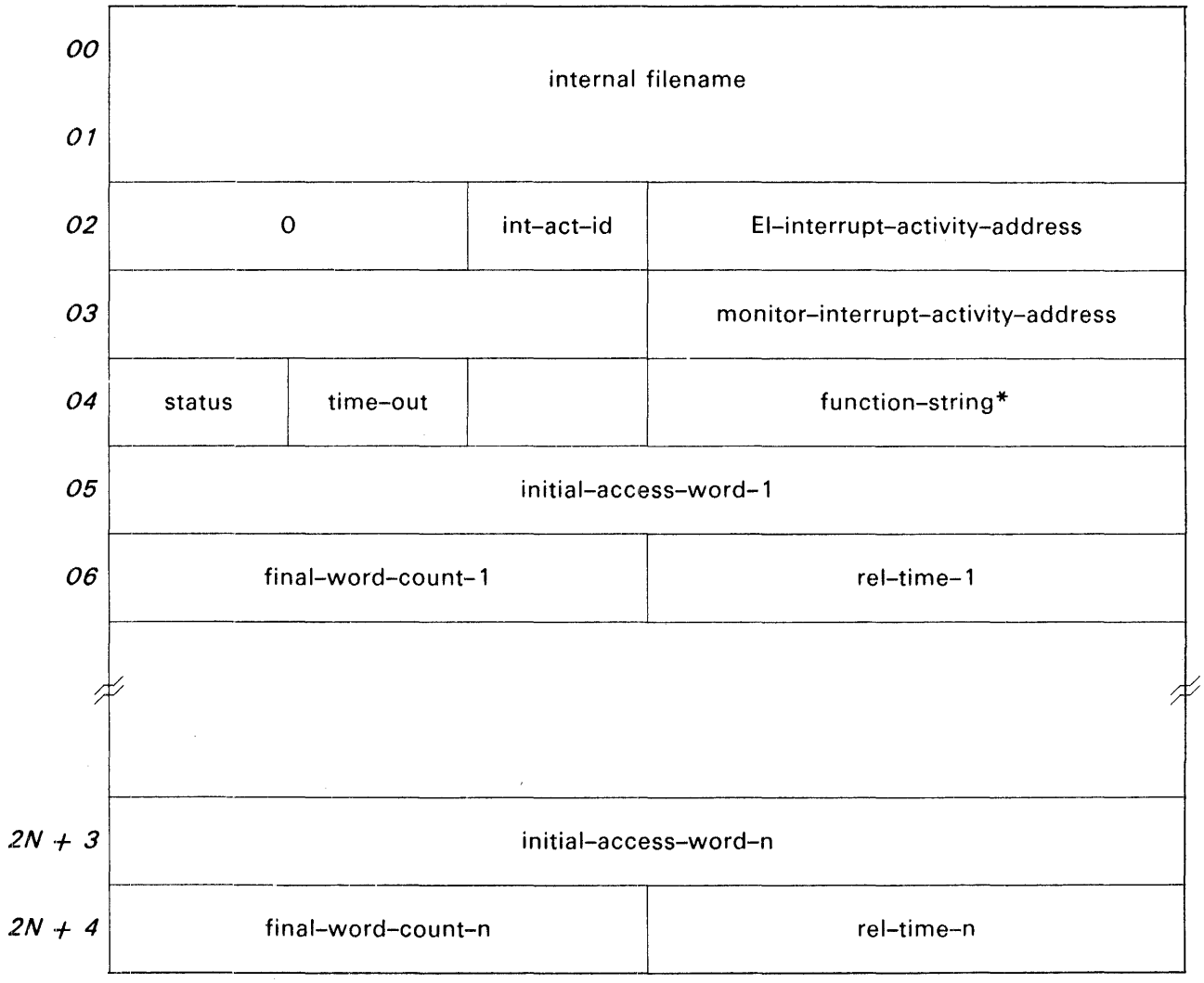
The arbitrary device interface (ADI) allows the user to directly control the I/O functions to a device on an I/O channel for special devices where standard interfaces are not provided, and for special operations on devices where standard interfaces are provided.

For all Series 1100 systems except the 1100/80, the interface is entered through either the IOARB$ or IOAXI$ requests described in 6.8.2 and 6.8.3, respectively.

6.8.1. Arbitrary Device I/O Packet

6.8.1.1. Packet Format

The format for the arbitrary device I/O packet as illustrated in Figure 6-3.



* See 6.8.6 if I/O path selection is to be used.

Figure 6-3. Arbitrary Device Packet

Word 0 and 1

The internal filename (see 2.6.2) is used in all references to the file. It is specified in Fieldata, left-justified and space filled. The unit must be placed in the reserved state by operator keyin.

Word 2

int-act-id — Numeric identity (1-35) given to the external interrupt activity (IOARB$ only). This field can be left as zero as for the standard handler packet if no synchronization is required with this activity. For entrance at IOAXI$, the interrupt activity has the same activity-id as the original activity.

EI-interrupt-activity-addr	Address at which control is given upon occurrence of an external interrupt or any status of 0-017. This activity address is always required.

Word 3

monitor-interrupt-activity-addr	The address at which the interrupt activity is given control if the function string indicates a monitor interrupt is to be returned to the user, and the interrupt which indicates completion of the operation is a monitor interrupt. This address is optional (see 6.8.1.3).

Word 4

status	Status code indicating the disposition of the request.
time-out	The number of dayclock intervals for which the subsystem is timed before the lack of a monitor or external interrupt is to be considered an error. The value 1 corresponds to 6-12 seconds, 2 to 12-18 seconds, and so forth for the 1106/1108 Systems. The maximum time indication is 77_8. For the 1110, 1100/10/20/40 Systems, the value 1 corresponds to 6.55-13.10 seconds, 2 to 13.10-19.66 seconds. If an operation is left outstanding on a channel for a time in excess of this value, a unique status code is returned to the packet.
function-string	Consists of a group of 3-bit bytes (octal digital string) interpreted from left to right (bits 17-15 compose the first byte). The assigned codes are: 0 - I/O path selection (see 6.8.6) if first byte, otherwise end of string. 1 - Initiate function mode without monitor (LFC) 2 - Initiate function mode with monitor (LFCM) 3 - Initiate output mode without monitor (LOC) 4 - Initiate output mode with monitor (LOCM) 5 - Initiate input mode without monitor (LIC) 6 - Initiate input mode with monitor (LICM) 7 - Wait for unsolicited interrupt (must be first octal digit)

Words 5,7,...,2N+3

Words 6,8,...,2N+4

The initial access words to be used to control the channel. See 6.8.6 if I/O path selection is to be used.

final-word-count	Final word count as contained in access control register.
rel-time	Relative time between execution of the corresponding operation in the string and the execution of the next operation or the occurrence of an interrupt. The time is given in 200-microsecond increments.

6.8.1.2. Function String Interpretation

Starting at the left of the function string, the operations represented by the code are carried out as directed. As the string is interpreted, succeeding pairs of access words are referenced. The final word count of the preceding operation is updated and the initial access word for the current operation is loaded. At most, six modes can be specified in the initiation string. As a practical limit, the combined length of all external function buffers is set at 9. Exceeding this count is considered a program logic error and causes reference to the error mode return point. As an example of string interpretation; if an input operation is to be performed with termination by an external interrupt, the initiation string could be $5\,10000_8$ with two sets of access words. The first operation is to load the input channel assigned to the filename specified in the packet using the access word in word 5. This is followed by a Load Function in Channel instruction (LFC), using the access word in word 7 to locate the function word. Upon occurrence of an external interrupt, the final access word count and the relative time are stored in word 6, and the final values for LFC are placed in word 8.

6.8.1.3. Monitor Handling

The user can specify instructions in any desired order to perform a particular I/O operation. When a monitor instruction is encountered, further interpretation of the string stops until the particular monitor occurs. Thus, the user program must make certain that the proper instructions are monitored to ensure that the respective access words do not get overlaid; that is, if two successive operations initiate output transfer, the first one should be with monitor unless the time between I/O instruction executions allows for transfer of all words of the first output buffer. To determine whether or not an access word has sufficient time to count down between initiation of operations and hence possibly allow operating at times without a monitor, the minimum time between execution of the I/O instructions is at least 10 microseconds (this varies upward, depending upon operation, overlapping data transfers, etc.). For such sequences as a function transfer of a single-word external function (EF) buffer followed by an output transfer, this is sufficient time for the function transfer to be completed before output transfer is initiated without the necessity of monitoring the function transfer.

The appearance of monitored modes does not necessarily indicate the need for a monitor completion activity (specified in word 3), as the interface interprets intermediate monitor modes. A monitor activity is required if either:

1. the last mode in a string is with monitor; or

2. the last mode is not monitored, and no external interrupt is expected to signal conclusion of the mode established as a result of the final mode.

If any monitored modes precede the final mode, a wait for external interrupt is done after the final I/O instruction is executed if the monitor-interrupt-activity-addr (word 3, $H2=0$) is not specified. No wait for external interrupt is done and control is returned to the monitor interrupt activity immediately if a monitor-interrupt-activity-addr (word 3, $H2\neq 0$) is specified. For example, an input drum operation is normally terminated without interrupt; hence, the sequence LFC, LICM, LFC is used, and a monitor interrupt activity is specified and executed without waiting after sending out the second function following the input monitor interrupt, whereas an output drum operation is normally terminated with interrupt; hence, the sequence LFC, LOCM, LFC may be used without a monitor interrupt activity, in which case a wait for external interrupt is done after sending out the second function.

6.8.1.4. Disconnecting Channel

Regardless of the manner in which the interface gives control to the interrupt activity, in all cases, the input and output active states are cleared on the particular channel by execution of the Disconnect Input in Channel (DIC) and Disconnect Output in Channel (DOC) instructions before control is given to the interrupt activity.

6.8.1.5. Function (EF) Handling

When a function mode is called, the interface inserts the proper unit designator and adds the proper base address to the relative address of the function word. At that time, if the channel contains equipment shared by other assignments, it may be necessary to perform certain error checks to prevent leaving the channel in an indeterminate state and to prevent intrusion upon other assignment privacy. Nonstandard special I/O devices are assigned by channel, and the interface makes no modifications to the function words for these devices.

The function buffer for magnetic tape or mass storage channels is limited to a word count of one word, except for search functions, in which case a second word, the identifier, is allowed. For other than these cases, in a multiple-word EF buffer, each word is modified by the unit designation and subjected to the particular tests based on equipment type.

6.8.1.6. Interrupt Activity Handling

Word 2 and word 3 of the packet may be used to specify interrupt activities, one of which is executed when the corresponding interrupt occurs. Word 2 specifies the activity to be executed in case of an EI. The lower half of the word gives the activity starting address, and S3 is set to the activity identity if synchronization is necessary. The register save and priority are assumed to be X8 through X11,A0 through A5,R1 through R3, and top priority, respectively. An EI activity must always be specified regardless of whether a monitor interrupt is to be used. The monitor activity is defined in the same format as the EI activity. If both a monitor and an EI occur, the EI activity is given control, and occurrence of the monitor interrupt can be determined by examining the access word. If an EI occurs after control is returned to the user because of a monitor, the EI is discarded. The EI is given priority over the monitor in order to give status to the user if the EI should occur first or simultaneously with the monitor. When control is given to the interrupt activity, register A0 is loaded with the packet address, and, for the EI activity, register A1 contains the EI status word. See 6.8.2 and 6.8.3 for specific characteristics of the interrupt activity associated with the ER's function string.

Upon completion of an I/O operation, a status code is stored in S1 of word 4 of the request packet denoting the conditions of the completion. See Appendix C for arbitrary device interface status codes.

6.8.2. Initiate ADI and Return Control Immediately (IOARB$)

Purpose:

Initiates an arbitrary device I/O operation with control returned, in line, as soon as the request is either listed or the operations have been initiated. An interrupt activity is initiated when the request is completed.

Format:

```
L,U  A0,pktaddr
ER   IOARB$
```

This may be generated by the procedure call:

```
I$OARB   pktaddr
```

Parameters:

pktaddr	Address of device I/O packet (see Figure 6–3).

NOTE:

IOARB$ is not supported on the 1100/80 System.

6.8.3. Initiate ADI and Exit with Interrupt (IOAXI$)

Purpose:

Initiates an arbitrary device I/O operation with the caller, simulates an exit function, and controls the return to the program at the appropriate interrupt activity specified in the request packet.

Format:

```
L,U  A0,pktaddr
ER   IOAXI$
```

This may be generated by the procedure call:

```
I$OAXI   pktaddr
```

Parameters:

pktaddr	Address of device I/O packet (see Figure 6–3).

Description:

The activity performing the IOAXI$ request does not actually exit, but saving and restoring registers is eliminated (except for register A0), and the register set is reduced to the minor set only. The continuation of the IOAXI$ activity at the interrupt point is with the same activity–id; hence, the value in the int–act–id field is ignored for the IOAXI$ request.

NOTE:

IOAXI$ is not supported on the 1100/80 System.

6.8.4. Free Format Disk Interface and MSA Tape ADI

The handler is designed to format, read, and write disk packs and MSA tapes in other than the standard Series 1100 Executive formats. It is an adaptation of the arbitrary device interface to control multi–interrupting, byte–oriented, command chain subsystems with the Series 1100 Executive operating on other drives of the same subsystem.

The I/O packet format for free format disk and MSA tapes is as described in 6.8.1 and illustrated in Figure 6-3 with the following exceptions.

Word 3

The monitor-interrupt-activity-addr field is unused. No monitor operations are permitted for disk or MSA tapes. When the packet is checked for the function string (word 4), any monitor operation that is encountered causes the program to be placed in error mode with an error type of 01 and a code of 025 (see Appendix C.3).

Word 4

The time-out field specifying the number of dayclock intervals is unused for disks. All disk I/O operations are confined to one 6-second time interval by the Executive System.

For disk or tape, an I/O request is limited to one external function (EF) access control word per packet where the function access word may be up to 11 words in length for disks, and two words in length for tapes. The EF access words may contain the command parameters as well as the command string. For example, an operation to do a disk read may be as follows:

Word 0 - Set file mask command
1 - Seek command
2 - Search command
3 - Read command
4 - Jump command
5 - Set file mask parameter
6-7 - Seek parameters
8-9 - Search parameters

When one function operation has been found and a second function operation is indicated, the program is considered in error and causes it to be placed in error mode with an error type 01 and code 021.

Once the mode string has been accepted and the I/O is to be started at initiation of the EF, a delay occurs if the next operation is to be an output operation. This delay is done to ensure that the EF chain and parameters access word has been sent before the output access control word is issued. Otherwise, the output access control word would overlay the EF access control word. If an input operation follows the issuance of the function chain, no delay occurs in opening input (LIC).

If an error is encountered by the MSA handler during an auxiliary status or sense request, the code is passed to the user in S1 of A3, plus the upper half of A1 is flagged with 0777777 to indicate that the abnormal condition occurred. If the error occurred on the sense request and if auxiliary status was also being obtained, the auxiliary status is placed in A2. If the error occurred on the auxiliary status request, the contents of A2 are unpredictable. The codes which may appear in S1 are:

077 Test function failed
076 Sense bytes were not received by processor
075 MSA error on sense request
074 Unit check on sense request
073 Interface parity error on sense request
072 Interface parity error on auxiliary status request
070 No path UP for interrupt received
000 See EI

Once the I/O access control word has been initiated by either a Load Input Channel (LIC) or Load Output Channel (LOC), another I/O access control word (ACW) may follow until the limit of six modes

is reached. This method is not advised because no delays are done after the ACW has been initiated, and thus the new ACW would overlay the previous ACW before it was completed. As a practical limit, it is recommended that a single I/O operation following the EF be used per packet request.

The EF command chain is checked to ensure that the M-field (multiple function string) of the MSA/disk command is not set. If the M-field is set, a program logic error is assumed which causes the program to be placed in error mode with an error type 01 and code 032.

The device address field of the MSA/disk command is cleared by the Executive and the file associated device address is inserted. A privileged user accessing control unit address F_{16} does not have the unit portion of the device address field modified.

The status information is obtained as follows:

- The external status received by the Executive is returned at the normal mode return point in register A1 where a secondary status is found in register A2. If during the operation, an error condition (MSA error, unit check) is detected by the Executive from the external status, the auxiliary status and sense bytes (bytes 0 through 23) are requested and returned with the original status at the normal return point. The external status occupies register A1 with the secondary status in registers A2, A3, A4, A5, R1, R2. The sense bytes are returned in the A Format; that is, Q1 = sense byte 0, Q2 = sense byte 1.

- When the external status specifies a busy status, the Executive waits for the control unit and, external status and upon receiving it, causes reference to be made to the normal mode return point with the statuses OR'd together in register A1.

- When an external status specifies a channel end without an accompanying device end, the Executive waits for the device end status and then causes reference to the normal mode return point with the device and external status OR'd together with the first status in register A1.

6.8.5. Programming Considerations for Using ADI with Tapes

The standard tape assign mechanism is used to assign a tape to be used via the arbitrary device interface. The user does not need to know the device address of the tape unit assigned since it is assigned internally by the Executive. Labeled tapes cannot be used with this interface.

For MSA tapes, status information is obtained and returned to the user as follows:

> In the event of an MSA error or unit check, the auxiliary status and sense bytes in addition to the EI are returned to the user. When the abnormal byte count bit is set, the auxiliary status is returned with the EI so the user knows the abnormal byte count (0-8). If the abnormal byte count is 0, the operation was a read, and the access word was counted down, then a short read was done.

The first EI is returned to the user in register A1; the auxiliary status (when returned to the user) is in A2, and the sense bytes are in quarter-word format in A3 (sense bytes 0-3) and A4 (sense byte 4 in the upper quarter). If the unit is busy, control is returned to the user with the busy bit set in the EI. When a channel end is received without a device end, the interrupt is put into A1 of the user's register save area, but control is not returned to the user until a unit completion interrupt (device end, unit check, status modifier, busy, or MSA error) is received or until the unit times out. At such time the second status is OR'd with the first status in A1 (except for the MSA auxiliary status in A2).

An explanation of the EI and the auxiliary status and the exact meaning of each bit may be found in UNIVAC 1100 Series Multi-Subsystem Adapter Programmer Reference, UP-7890 (current version). An explanation of the sense byte may be found in SPERRY UNIVAC 1100 Series UNISERVO

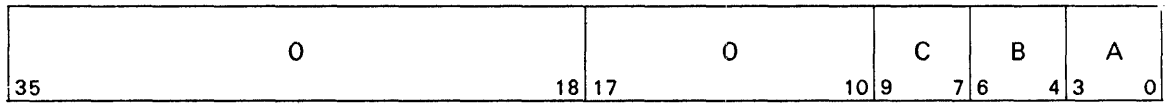
12/14/16/20/30/32/34/36 Magnetic Tape Subsystems, Programmer Reference, UP-7943 (current version).

6.8.6. I/O Path Selection via the Arbitrary Device Interface

A program to specify the I/O path to be used to communicate with a peripheral device. The only restriction is that the specified path must have been configured as connected to the referenced device. If the specified path is not configured as connected to the device, the program is placed in error mode with an error type of 01 and a code of 025 (see Appendix C.3). A privileged user may access a downed control unit via I/O path selection.

In order to direct I/O to the device over a specific channel, the function string in the I/O packet (Figure 6-3) must obtain a zero as the first function. A zero anywhere else in the string serves as an end-of-string and terminates processing of the function string. The path to be used is specified in word 5 of the packet. The format of word 5 differs for 1106, 1108, 1100/10/20 and 1110, 1100/40 Systems as described below:

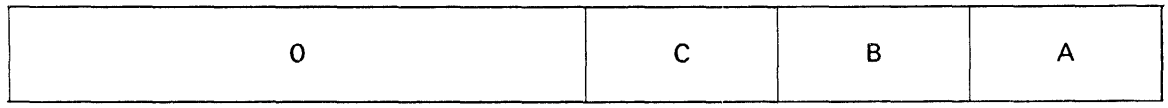
Format for 1106, 1108, 1100/10/20:



where:

A = CPU channel number
B = CPU number
C = IOC or MSA channel number, if applicable, or 0

Format for 1110, 1100/40:



where:

A = IOAU channel number
B = MSA channel number, if applicable, or 0
C = CAU number

An example of an I/O packet specifying path selection is:

00	internal filename			
01				
02	0		int-act-id	interrupt-activity-addr
03				monitor-interrupt-activity-addr
04	status	time-out		016100 function-string
05	desired path specification			
06	initial-access-word-pair			
07				

The left-most zero in the function string specifies path selection. The next zero specifies end of function string.

6.8.7. Auxiliary Storage Interface via the Arbitrary Device Interface

A user program can interface directly with a mass storage unit. The program must have previously assigned the entire unit using the specific unit assign capability (see 3.7.1.3.2).

The unit to be assigned must be in the reserved state (via RV operator key-in) and not assigned to another program.

The arbitrary device interface inserts the correct unit number in the appropriate external function (EF) words so the user program need only provide a unit relative address of the area to be referenced.

When interfacing with auxiliary storage and a continuous read or write function is specified, a terminate EF should be specified after the LICM or LOCM command in the string. Also, when executing the command string, string execution stops if a monitor command (i.e., LFCM, LICM, or LOCM) is encountered. Thus, when setting up to perform a read operation on drum, the command string should be LFC(1), LICM(6), LFC(1), EOS(0). The final LFC should be a terminate without interrupt function (23_8), if a monitor interrupt activity is specified; or a terminate with interrupt function (33_8) if not specified.

If a terminate with interrupt function is issued as the last command in the string, a monitor activity must not be specified if the value of the EI status word is required. This is true only if a monitor interrupt was requested in a prior command in the string.

6.9. ARBITRARY DEVICE INTERFACE (1100/80)

The Arbitrary Device Interface (ADI) provides the ability to handle standard SPERRY UNIVAC-supplied devices in a user-specified manner, as well as to handle Arbitrary Device (ARBDEV) configured peripherals. It allows the user to directly control the I/O functions to a device on an I/O channel for special devices where standard interfaces are provided.

For 1100/80 Systems, the interface is entered through the IOADH$ request.

Format:

```
L,U  A0,pktaddr
ER   IOADH$
```

This may be generated by the procedure call:

```
I$OADH    pktaddr
```

Parameters:

pktaddr — Address of device I/O packet (see 6.9.1).

6.9.1. Arbitrary Device I/O Packet Format

Access to the Arbitrary Device Handler is via a ER to IOADH$, with A0 containing an ADI packet address. The ADI packet has the following format:

<table>
<tr><td>00</td><td colspan="4">internal filename</td></tr>
<tr><td>01</td><td colspan="4"></td></tr>
<tr><td>02</td><td colspan="3">unused</td><td>interrupt activity address</td></tr>
<tr><td>03</td><td>packet status</td><td>function code</td><td colspan="2">unused</td></tr>
<tr><td>04</td><td colspan="2">device end time</td><td>forced path selection indicator</td><td>forced path selection path</td></tr>
<tr><td>05</td><td colspan="3">length of device status buffer</td><td>address of device status buffer</td></tr>
<tr><td>06</td><td colspan="3">number of CCWs</td><td>address of CCW list</td></tr>
</table>

Figure 6-4. Arbitrary Device Handler Request Packet

Words 0 and 1

internal filename

This filename must have been assigned using the whole unit assignment, i.e.,

@ASG Filename,*Device Name,Parameters.

In the case of tapes, any method of assignment is acceptable.

Word 2

interrupt activity address

If zero, control is returned at the address of the ER plus one. All registers remain intact. If the interrupt activity address is specified, then control is returned to this address at the completion of the request.

Word 3

packet status:

Upon completion of an ADH request, the packet status reflects the results of the request. Bit 35 must be zero when making the I/O request.

00 – Interrupt status has been received. The caller must determine the validity of it.

01 – the device has timed out

05 – PCI bit set in CCW or data format changed from one CCW to the next in a data chain.

07 – the CCW list appeared to complete in an abnormal manner – check the device status buffer for the exact reason.

012 – CCW quantity exceeded maximum of 254 allowed

014 – Requested path not available.

015 – CCW data count equal to zero or invalid CCW format bits.

016 – EF quantity exceeded the maximum of eight allowed for a nonarbitrary device

017 – EXPOOL not available for some part of the request. User may exit or retry after an appropriate wait.

020 – Invalid function specified via packet

021 – the file name specified in the ADH packet is not assigned to this run.

022 – Reference attempted beyond the assigned file when the file is configured as a FH-432 or FH-1782 drum.

023 – The ADI packet is not within program limits

024 – File is not accessible via ADI

025 – Data or EF buffer not within user write enabled storage.

026 – The interrupt activity starting address is out of program limits

027 – The ADI packet specified by the address in A0 is already in use by another ADI or IO request. (Most significant bit of 6-bit packet status was set at the time ADI was called).

030 – Device status buffer not within user write enabled storage

031 – CCW quantity equal to zero for function 1.

032 – CCW not within user storage.

034 – data address portion of at least one of the CCWs in the CCW list

035 – PCI was specified in CCW list.

function code

001 – issues the CCW list to the indicated device and reports the hardware status to the requestor.

002 – puts the requestor in a Wait for Unsolicited Interrupt queue. If the assigned device provides an unsolicited interrupt, the requestor gets control at the supplied EI address, if it exists, or the location following the ER instruction.

Word 4

device end time

This value specifies the number of 6-second intervals that may occur before the final interrupt is received. If this interrupt is not received within this time, the device has timed out and control is returned with the appropriate status in the ADI packet.

forced path selection indicator

0 – system selects path to device

1 – forced path selection path contains physical path with which to address the service

forced path selection path

If forced path selection indicator is equal to 1, forced path selection path must contain the device path. The path must be configured.

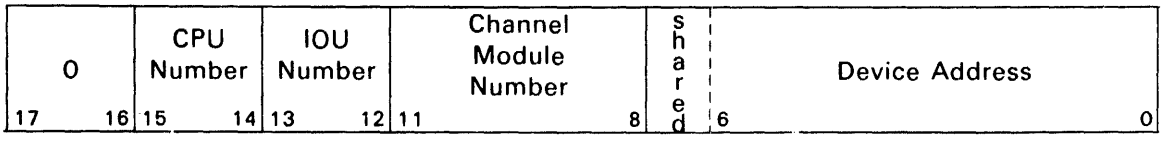


Figure 6-5. User-Specified Path Format

Bit 7 — 0 = nonshared
1 = shared

Word 5

length of device status buffer — Variable from 0 to n based on amount of information desired by user.

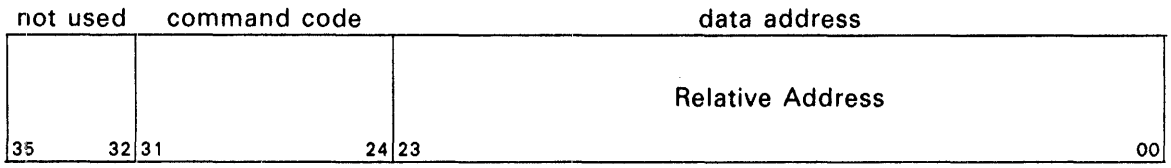
address of device status buffer — See 6.9.2 for format and usage of device status buffer.

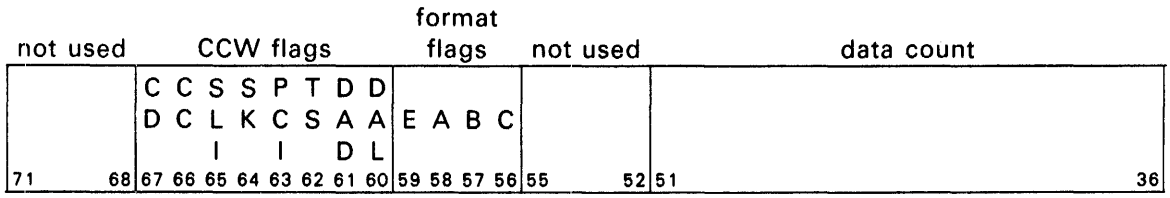
Word 6

number of CCWs

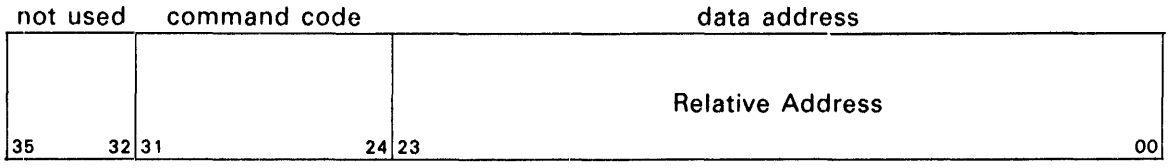
Number of Channel Command Words (CCWs) in the CCW List (maximum of 254 allowed). The CCWs have the following formats:

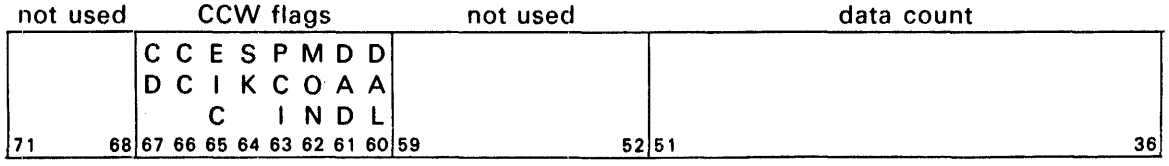
Byte or Block Multiplexor Channel CCW





Word Channel CCW





where:

Channel Command Word

A channel command word specifies the command to be executed, and for commands initiating I/O operations it designates the storage area associated with the operation, the amount of data to be transferred, the formatting of data that is to be done, and the action to be taken when the operation is completed.

The fields in the CCW are allocated for the following purposes:

Data Address — Bits 00–23 contain the storage address of the first data word to be transferred unless the command code is Transfer in Channel or Store Subchannel Status. For the Transfer in Channel command, the field contains the new CCW address, and for the Store Subchannel Status command the field contains the address where status is to be stored.

Command Code — Bits 24–31 specify the operation to be performed by the subchannel and device.

Byte or Block MUX Channel Command	Code	Word Channel Command
Invalid	XXXX(1) 0000	Invalid
Sense	MMMM(2) 0100	Invalid
Transfer in Channel (TIC)	XXX0(1) 1000	Transfer in Channel (TIC)
Store Subchannel Status	XXX1(1) 1000	Store Subchannel Status
Read Backward	MMMM(2) 1100	Invalid
Write	MMMM(2) MM01	Write
Read Forward	MMMM(2) MM10	Read
Control	MMMM(2) MM11	Forced External Function

CODES:

(1) X = Don't Care

(2) M = Word Channel; Don't Care. M = Byte or Block MUX Channel; Modifier.

On a byte or block MUX channel, commands that initiate I/O operations (write, read, read backward, control, and sense) cause all eight bits of the command code to be transferred to the I/O device. The modifier bits specify to the device how the operation is to be performed.

Data Count — Bits 36–51 specify the number of bytes to be transferred on a byte or block MUX channel, or the number of 30 or 36 bit words transferred on a word channel ISI interface, or the number of quarter words or half words transferred on a word channel ESI interface.

Special Flags — Bits 60–67 contain flags that specify data formats, special handling of an operation by the channel, and the action to be taken once the present operation is completed.

Chain Data (CD) — Specifies that upon completion of the portion of a data transfer operation being controlled by the current CCW, a new CCW is to be read from storage and the operation is to be continued under control of the new CCW.

Chain Command (CC) — Specifies that upon completion of the operation, a new CCW is to be read from storage and the operation specified by the new command code is to be initiated. If the Chain Data flag is set, the Chain Command flag is ignored.

Suppress Length Indication (SLI) — (Byte and block multiplexor channels only) – Specifies that if command chaining conditions are present, a command chain operation be initiated regardless of the residual byte count. The absence of the Suppress Length Indication bit specifies that if command chaining conditions are present, a command chain operation be initiated only if the residual byte count equals zero. If the command chaining conditions are present, and the Suppress Length Indication is not set, and the residual byte count does not equal zero, the execution of the CCW list is terminated, subchannel status is generated, and an interrupt is presented to the processor. The Suppress Length Indication flag is ignored if the Truncated Search flag is set in the same CCW.

NOTE:

Command chaining conditions are defined as channel end and device end status, the command chain flag set, and the data chain flag clear in the active CCW.

EI Chain (EIC) — (Word channel only and ESI subchannels only) – Specifies that upon completion of the operation at the ESI word channel device, a new CCW whose storage address is four higher than that of the current CCW is to be read from storage and the operation specified by the new command is to be initiated. The external interrupt presented by the device is stored in the status table prior to chaining, and a tabled interrupt request is presented to the processor.

The external interrupt chain is not executed if:

1. the status table subchannel is not active;
2. a hardware error is detected when entering the external interrupt in the status table;
3. a hardware or software error is detected during the retrieval of the new CCW from storage; or,
4. the subchannel was not in an active state before receiving the external interrupt.

The EI chain flag is not interpreted on ISI word channels.

Skip Data (SK)	Specifies that data is not written in storage for input operations. However, the device and subchannel are handled in the same manner as during conventional input operation. For output operations, the Skip Data Flag is ignored.
Program Controlled Interrupt (PCI)	Specifies that the channel shall store a PCI subchannel status indication and generate an interrupt as soon as possible after a CCW containing this flag is obtained.
Truncated Search (TS)	(Block multiplexor channel only) – Specifies that a special chaining operation is to be executed. The channel saves the command in the CCW with the truncated search flag and also saves the command from the preceding CCW. These two commands are then reissued to the control unit during a truncated search operation. See 8.4 for a detailed description of truncated search operations.
Monitor (MON)	(Word Channel Only) – Specifies that the channel shall store subchannel status and generate an interrupt when the data count in the final CCW has been exhausted.
Data Address Decrement (DAD)	Specifies that the data address be decreased by one for each *full data word* transferred under control of the current CCW. This flag is ignored if the Data Address Lock (DAL) flag is set. If neither the DAD or DAL flags are set, the data address is incremented by one for each full data word transferred.
Data Address Lock (DAL)	Specifies that the contents of the data address field remain unchanged for each word transferred under control of the current CCW.
Emulation Mode (E)	(Byte and block multiplexor channels only) – Specifies either 1100 mode or 494 mode for data transfer operations. If E is zero, the Series 1100 mode is selected and the 36-bit data packing format is selected. If E is one, the 494 series (Emulate) mode is selected, and the 30-bit data packing format is selected. If E equals one (Emulate mode), block multiplexor channel format B and format C input data transfers are 36-bit full word writes with bits 30 through 36 zero filled. On word channels the E-bit is ignored and the mode of operation (494 mode or 1100 mode) is specified by patch wire. Each word channel Series 1100 ESI interface operates via patch wire in either quarter-word mode or half-word mode. On a word channel 494 series ESI interface, input data transfers are 21-bit partial word writes with bits 15 through 29 of the data word containing the input character and bits 30 through 35 being zero filled.
Format Flags (A, B, and C)	(Byte and block multiplexor channels only) – Specify the packing format of data bytes. Format A or Format B must be selected on the byte MUX channel. The format flags are ignored on a word channel. The contents of bit positions 32–35, 52–55, and 68–71 of the CCW is ignored.

6.9.2. Device Status Buffer (DSB)

The device status buffer is a user-supplied buffer that ADI uses to return hardware information related to the outcome of the ADH request. The format of the DSB is as follows:

Table 6-7. Device Status Buffer

00	number of sense bytes returned	software condition code		user relative address of last CCW executed
01	initiation time			
02	completion time			
03	CSW0 (External Interrupt)			
04	CSW1 (MSA auxiliary status)			
05	CSW2 (EI status word)			
06	currently not used			
07	sense bytes 0 – 3			
010	sense bytes 4 – 7			
≈				
n				

where:

number of sense bytes returned — Number of sense bytes stored in the device status buffer

software condition code — Software condition code returned:

- 0 – I/O appeared to complete normally.
- 1 – condition code of 1 received
- 2 – condition code of 2 received
- 3 – condition code of 3 received

4 – time out
6 – CSW contained residual data count.
7 – CSW did not contain address of last CCW in the CCW list+2.
10 – channel status error (CSW status not equal to channel end/device end)
11 – Unit check but no sense bytes

user relative address of last CCW executed	User relative address of last CCW executed calculated from address portion of CSW1. Set to zero if no correlation can be made between the actual CCW address and the next CCW address returned in a CSW.
initiation time	Value in dayclock at time I/O start requested by ADI
completion time	Value in dayclock at time control returned to ADH after I/O completion.
	The remainder of the DSB is variable and the user must perform various tests to determine what information, if any, is presented there. IOHADH returns as much information as possible to the user; i.e., if an ADI request is made with a DSB length of 3, then only those three words are completed.
CSW 0	First word of the CSW. Not valid for software condition codes 2, 3, and 4.
CSW 1	Second word of the CSW. Not valid for software condition codes 2, 3, and 4.
CSW 2	Third word of the CSW (EI Status Word). Not valid for a byte channel or a software condition code of 2, 3, or 4.
Words 7 thru n	Sense bytes returned in quarter-word format starting with Sense Byte 0 in the most significant quarter-word of Word 7.

6.9.3. General Information and Restrictions

1. The PCI bit may not be set; if set, error status = 035.

2. Since the user CCW list is transferred into EXPOOL before it is issued, the address portion of CSW 0 is an EXPOOL address, therefore, of no use to the user.

3. 254 CCWs is maximum number allowed.

4. On word channels, all noninterrupting functions should be chained to a terminate command or use the monitor bit in the CSW.

5. In the event that the device splits status, the first word of the CSW (CSW 0) and the residual data count of the second word (CSW 1) are those values presented in the CSW of the initial interrupt (normally device Channel End status). The device and subchannel status field of the second word are the result of ORing the values from the initial and terminal CSWs.

6. Since the user's CCW list is moved to EXEC storage prior to I/O execution, the convention must be followed whereby any TIC command CCW which may be skipped by the presentation of a Status Modifier/Channel End status during a command chain may only TIC to a CCW which has already been encountered within the chain.

7. Since the type of channel (byte or block) on which the I/O is to be performed is not readily ascertainable by ADI, a change in the data format bits from one CCW to the next within a data chain results in the user being returned an error code 05.

8. If the device is configured on a word channel and it is not configured as an arbitrary device, the maximum number of EF words that a given CCW list may issue is limited to 8. The EF words are moved to EXEC storage and the appropriate unit number is inserted in them prior to I/O execution if the device is not configured as arbitrary. If the equipment is a 5046 Control Unit, the unit number is inserted only in those EF words in which the most significant bit is set.

6.9.4. Initiate ADI to Wait for Unsolicited Interrupt (Function = 2)

Purpose:

To register a request to Wait for Unsolicited Interrupt for an assigned device.

Description:

This type of request does not cause any I/O operation to be initiated, but rather registers a Wait for Unsolicited Interrupt activity to be activated by an unsolicited interrupt (e.g., attention interrupt) from the assigned device. If an unsolicited interrupt or a timeout condition occurs for the specified device, the activity is given control at the specified EI address. If an EI address does not exist, control is returned to ER address + 1. Since no I/O operation is performed, any CCW list contained in the packet will be ignored.

6.10. STATUS CODES

Upon completing an I/O request, a status code from 0 to 037 is stored in S1 of word 3 of the I/O packet indicating the status of the completion. All status codes from 020 through 037 are error conditions that cause an activity to be terminated or given control at its error contingency point (see 4.9.3). If an interrupt activity was specified by the requesting activity or one was created for it, the interrupt activity receives control at its error contingency point.

When an I/O Executive Request is initiated, the packet address specified in A0 is validated in two steps. First, the Executive checks the address to determine if it is within the requestor's storage area. If it is not, the requesting activity is placed in the error mode with an error type 4 and code 2. Next, I/O control validates the packet address for such things as write protection. If an error is found, the requesting activity or the interrupt activity (if the request caused one to be created) is placed in the error mode with a type 1 and code 023. The 023 status code (illegal packet address) is never returned in the I/O packet. See Appendix C for a complete list of I/O status codes.

7. File Control

7.1. INTRODUCTION

The file control routines exercise centralized control over all files in the system. The primary functions of these routines are to:

- Maintain a master file directory of both the cataloged and temporary files.
- Control mass storage allocation as new files are assigned and existing files are expanded.
- Provide the interface between the user programs and mass storage I/O device handlers (maintain a record of the absolute addresses of the file granules).
- Prevent assignment of or access to any exclusively assigned file by any run other than the run to which the file has been exclusively assigned.
- Automatically move files from mass storage to magnetic tape as available mass storage space becomes exhausted (rollout), and automatically retrieve these files when needed (rollback).
- Provide the means which enables the user to obtain information on the assignment, contents, and locations of a file.

7.2. FILE ORGANIZATION

7.2.1. Master File Directory

For files which are to be retained beyond run termination, entries in a master file directory (MFD) are constructed containing the identification and characteristics of each file and are maintained by the system. The process of entering a file in the master file directory is called cataloging and is effected by the @ASG and @CAT control statements.

The information contained in each file entry includes the following:

- External name of the file (see 2.6.2)
- Project identity from the @RUN control statement

- Account number from the @RUN control statement
- Date on which the file was cataloged
- Activity of the file, including the date of last reference
- Usage authorization
- Recording mode (magnetic tape only)
- Granularity and number of granules assigned (mass storage only)
- Number of reels of tape and tape reel numbers (tape only)
- Linkage to the granule description (mass storage)
- Pack-ids and number of packs (disk only)
- Backup information - SECURE tape reel(s), file number, etc.

7.2.2. Mass Storage Allocation

Mass storage is accessed by the Executive in two ways:

1. Sector format - A sector formatted file is accessed by sectors (28 words in length) which are allocated in granules of tracks or positions. A track is 64 addressable sectors of 28 words each or 1792 words of storage. A position is 64 tracks (4096 addressable sectors or 114,688 words).

2. Word-addressable format - A word-addressable file is accessed on a word basis but is allocated in the same manner as sector-format.

When a mass storage file is initially assigned, only the number of granules initially requested in the @ASG control statement are allocated. After that, only those granules needed to service a given expansion request are automatically assigned. When the expansion request is beyond the highest track currently assigned, a minimum of four tracks are allocated if still within the file's maximum size. For example, if the initial request was for one track and this was followed by a five track write to track 28 and a one track write to track 35, then the file would have tracks 1, 28-32, and 35-38 allocated. Tracks 2-27 and 33-34 are not allocated until reference is made to those tracks.

The file supervision routines automatically assign additional increments of mass storage space as required to satisfy the needs of the worker programs. The space availability function also handles the release of granules giving them available status. Release of any part of a granule causes the release of the entire granule area.

Since files can be released a granule at a time, it is possible to have an empty file cataloged in the system with master file directory items and no allocated space.

7.2.3. File Addressing

As an extension to the master file directory, the Executive maintains tables specifying the location of the various granules allocated to a given filename. These tables are stored in sector-sized areas of mass storage and are used by the device handlers to convert the relative location furnished in the request to absolute hardware locations. For example, a request to read at address 129 of a file with sector addressability would refer to the second sector of the third track assigned to the file. This reference table allows voids and various types of mass storage within a file.

Unsolicited console input messages are available to control mass storage availability. This allows mass storage to be taken from and returned to the configuration without forcing an initial boot which would cause all user files to be deleted from the system. It allows users to reserve units for their own assignments and allows arbitrary device handler access to those units.

Mass storage is defined to be in one of four states. They are:

UP – Up status indicates that a mass storage unit is fully accessible by the operating system. This status allows the Executive to read, write, and allocate on this unit.

SU – Suspend status indicates that a mass storage unit is accessible for I/O; however, the Executive cannot allocate any space on this unit.

RV – Reserve status indicates that a mass storage unit is accessible only by absolute assignments and arbitrary device handler I/O requests.

DN – Down status indicates that a mass storage unit is not to be used for any purpose by the system.

7.2.4. Exclusive Use of Files

The file supervisor routines allow assignment of mass storage files to any number of runs at one time providing the exclusive use option is not on the @ASG control statement. This option delays the assignment of a file until no other run has that file assigned to it and ensures that other runs are delayed until a run releases the needed exclusively-assigned files.

All magnetic tape files are exclusively assigned, regardless of the presence or absence of the option.

The read-and-lock and unlock functions are available at the I/O level where logically contiguous areas (successive relative addresses) can be exclusively assigned to allow other runs simultaneous access of all the unlocked portion of the file. The complete definition of the various functions involved and the timing limits to be considered are in Section 6.

7.2.5. Rollout and Rollback of Files

Depending upon the amount of available mass storage, the degree of use given to cataloging files on mass storage, and the manner in which files are assigned, the Executive may need to obtain additional space on mass storage by rolling out cataloged files to magnetic tape. This feature is provided automatically by the Executive. The points at which rollout is turned on and off are system generation parameters.

Rollout to magnetic tape occurs when the request for allocation reduces the available mass storage below a threshold specified at system generation.

The rollout routine utilizes a file's activity as part of the criterion for file rollout eligibility.

A request to assign a rolled out file causes the Executive to request mounting of the proper magnetic tape unless already mounted, to read in the file, and to automatically return the file back to mass storage. (The SECURE processor actually performs the rollout and rollback, see Volume 3, Section 7.)

7.2.6. Retrieving Facility Assignment (FITEM$)

Purpose:

Provides a method to obtain information on file or facility assignments.

Format:

```
LA  A0,(pkt-lngth,pktaddr)
ER  FITEM$
```

Description:

An internal filename (left-justified and space filled) must be placed in the first two words of the information packet.

The remaining words of the packet are filled as a result of the FITEM$ request.

The minimum packet length is nine words; the maximum packet length is dependent upon the equipment type:

Equipment	Length
Word-addressable mass storage, arbitrary devices	10
Communications devices	9
Sector-formatted mass storage	10
Magnetic tape, removable disk	13

If the pkt-length is 0377777_8, the maximum amount of information allowed for the equipment type is transferred to the packet. If the pkt-length given is less than nine or greater than the maximum for the equipment type, only nine words are transferred to the packet and an error status is returned in register A0 (see following).

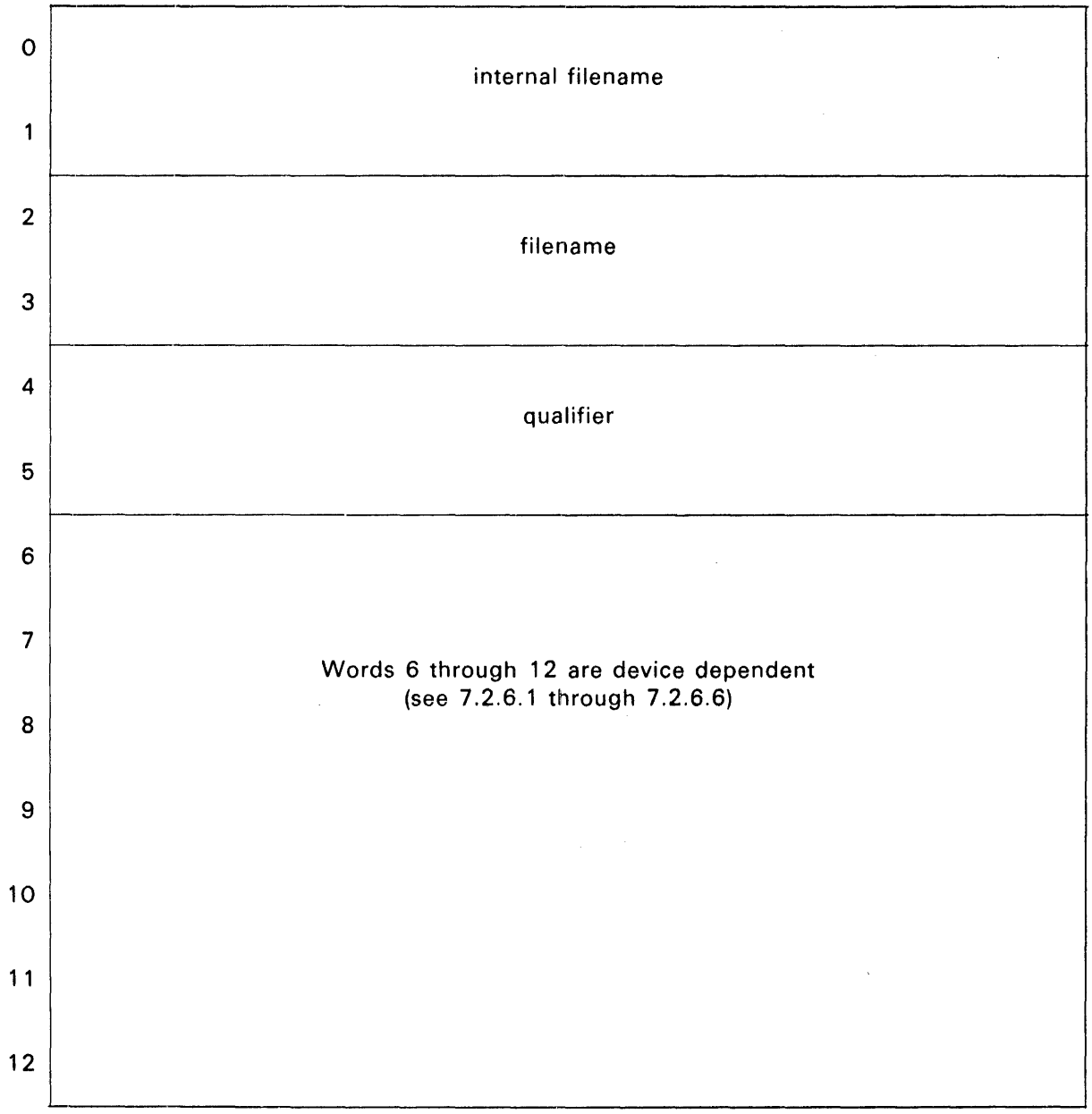
Rejection of the FITEM$ request occurs only if the relative packet address specified in the request packet is invalid; that is, the address falls outside the user's bounds, or the span of the FITEM$ packet violates the user's bounds, or the filename specified was not assigned to the run. If an invalid filename is encountered, the equipment type cell is zeroed.

The status codes (returned in S1 of register A0) applicable to FITEM$ requests are:

1_8 – The requested packet length exceeded the allowable maximum.

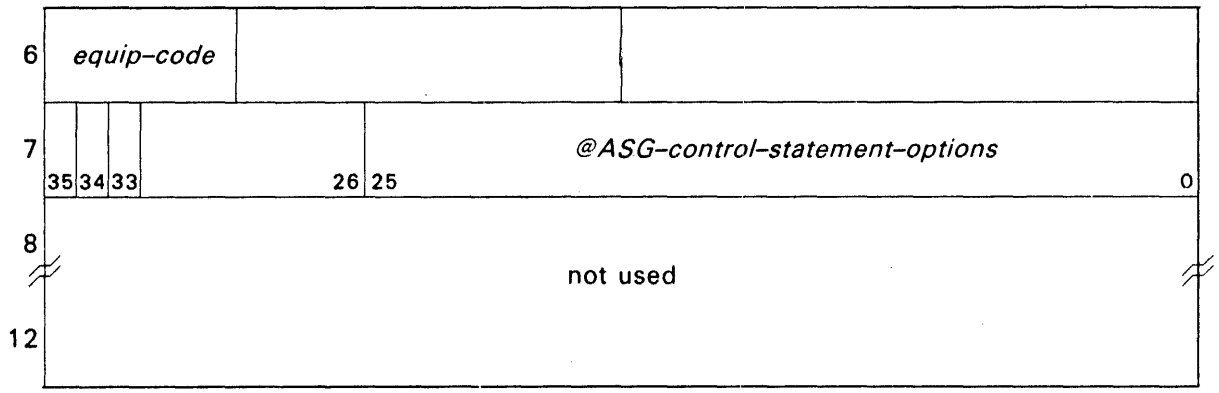
2_8 – The requested packet length was less than the allowable minimum.

Pktaddr is the address of a packet whose general format is:



7.2.6.1. Unit Record and Nonstandard Peripherals

The FITEM$ request packet format is:



Word 6

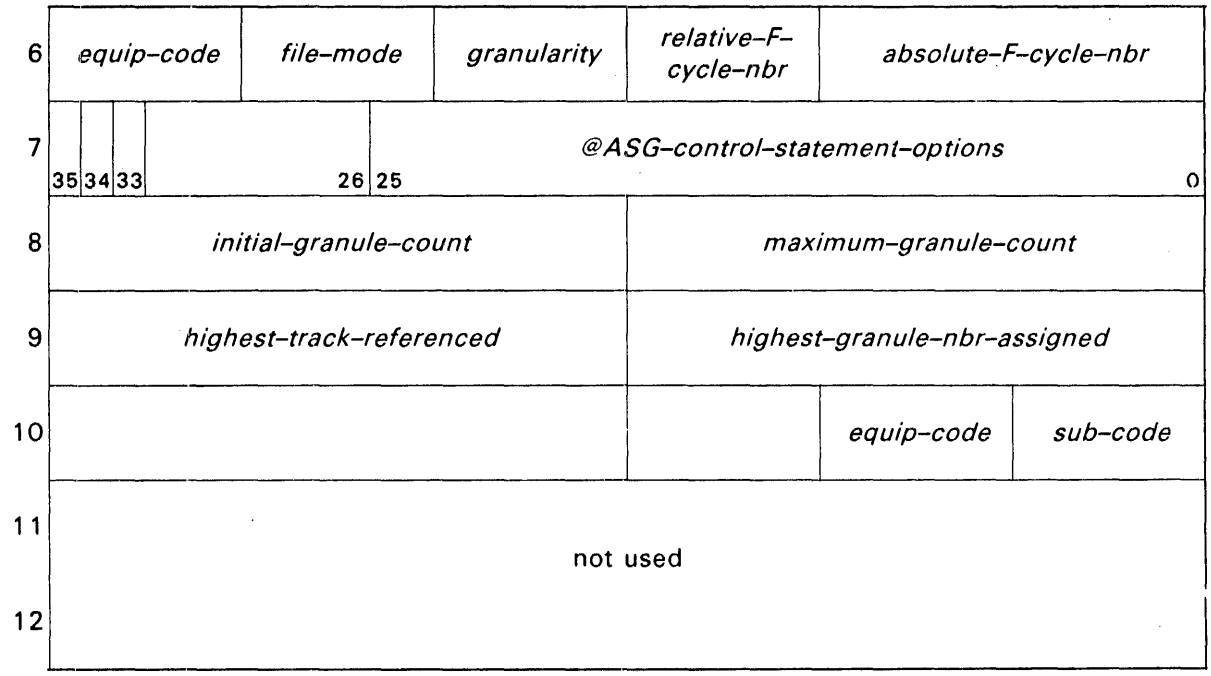
equip-code	Equipment code of the assigned facility. (See Table 7-1.)

Word 7

bit 35	If set, system has tape label checking turned on.
bit 34	If set, file assigned as a temporary file.
bit 33	If set, internal name is a use name.
@ASG-control-statement-options	Indicates the options specified on the @ASG control statement (see 3.7.1) assigned the equipment. Master bit notation is used; that is, a 1 in bit 25 indicates the A option was specified, a 1 in bit 24 indicates the B option was specified, and so forth.

7.2.6.2. Sector-Formatted Mass Storage Files

The FITEM$ request packet format is:



Word 6

equip-code	Equipment code of the assigned facility. (See Table 7-1.)
file-mode	Bit 29 set - Exclusively assigned file Bit 28 set - Read key is needed Bit 27 set - Write key is needed Bit 26 set - Read-only file Bit 25 set - Write-only file
granularity	All zeros indicate track granularity; nonzero indicates position granularity.

Word 7

bit 35	If set, system has tape label checking turned on.
bit 34	If set, file assigned as a temporary file.
bit 33	If set, internal name is a use name.
@ASG-control-statement-options	Indicates the options specified on the @ASG control statement (see 3.7.1) that assigned the equipment. Master bit notation is used; i.e., a 1 in bit 25 indicates the A option was specified, a 1 in bit 24 indicates the B option was specified, etc.

Word 9

highest-granule-number-assigned — The highest numbered granule (track or position) assigned to the file. This granule is the relative position of the granule within the file.

highest-track-referenced — The highest numbered track (track only) which has been written

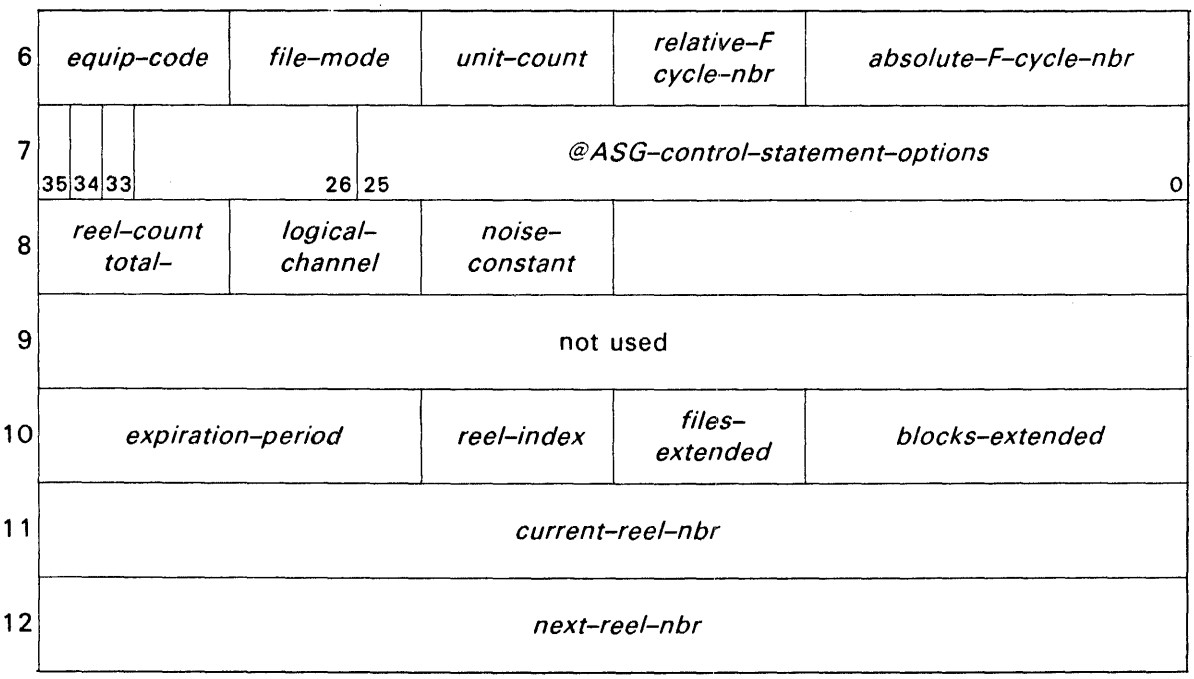
Word 10

equip-code — Equipment code of last facility upon which allocation occurred or of the facility containing the (cataloged) file's directory information or zero. (See Table 7-1.)

sub-code — Sub-code of the above equipment type. (See Table 7-1.)

7.2.6.3. Magnetic Tape Peripherals

The FITEM$ request packet format is:



Word 6

equip-code — (See Table 7-1.)

file-mode — Bit 29 set - Exclusively assigned file (always set)
Bit 28 set - Read key is needed
Bit 27 set - Write key is needed
Bit 26 set - Read-only file
Bit 25 set - Write-only file

unit-count — Number of units assigned

Word 7

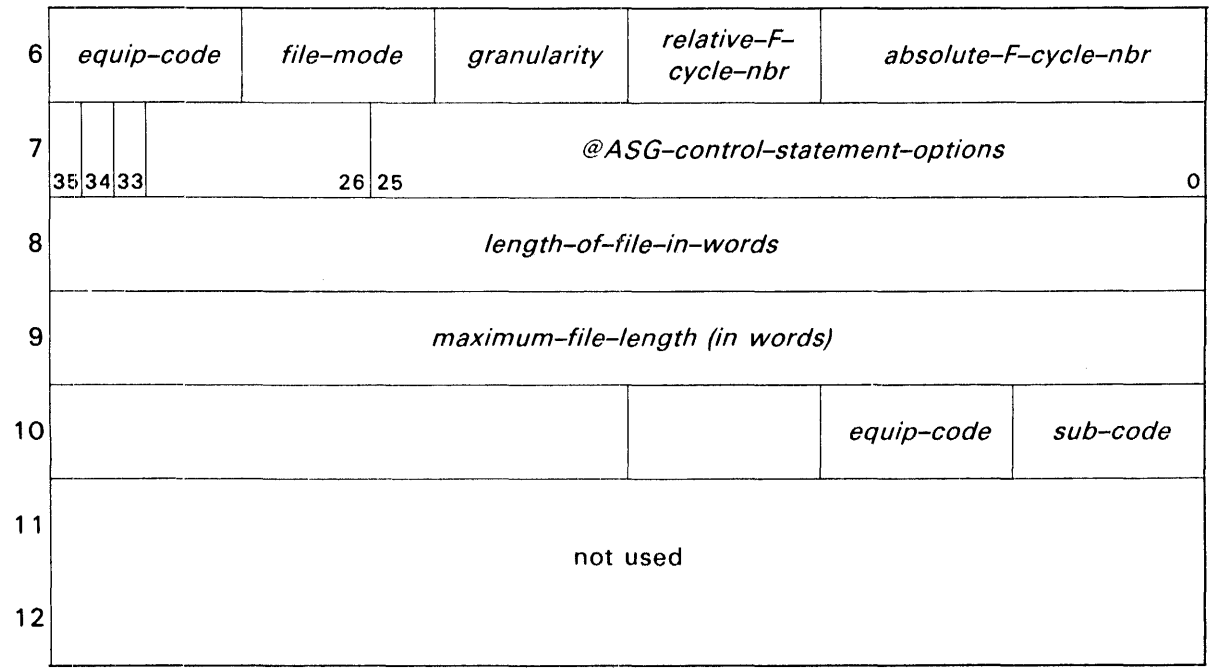
bit 35	If set, system has tape label checking turned on
bit 34	If set, file assigned as a temporary file
bit 33	If set, internal name is a use name.
@ASG-control-statement-options	Indicates the options specified on the @ASG control statement (see 3.7.1) that assigned the equipment. Master bit notation is used; i.e., a 1 in bit 25 indicates the A option was specified, a 1 in bit 24 indicates the B option was specified, etc.

Word 10

expiration-period	Specifies the number of days the file is to be retained (see 3.7.1.2).
reel-index	Index to the current reel of a multireel file
files-extended	Count of the number of hardware EOF marks encountered (applicable only to UNISERVO 12/14/16/20/30 tape units)
blocks-extended	UNISERVO 12/14/16/20/30: Count of the number of physical tape blocks encountered since load point or last EOF mark All other tape units: Count of the number of physical tape blocks encountered since load point.

7.2.6.4. Word-Addressable Mass Storage Files

The FITEM$ request packet format is:



Word 6

equip-code	Equipment code of the assigned facility. (See Table 7-1.)
file-mode	Bit 29 set - Exclusively assigned file Bit 28 set - Read key is needed Bit 27 set - Write key is needed Bit 26 set - Read-only file Bit 25 set - Write-only file Bit 24 set - Word addressable file (always set)
granularity	All zeros indicate track granularity; nonzero indicates position granularity.

Word 7

bit 35	If set, system has tape label checking turned on.
bit 34	If set, file assigned as a temporary file.
bit 33	If set, internal name is a use name.
@ASG-control-statement-options	Indicates the options specified on the @ASG control statement (see 3.7.1) that assigned the equipment. Master bit notation is used; i.e., a 1 in bit 25 indicates the A option was specified, a 1 in bit 24 indicates the B option was specified, etc.

Word 8

length-of-file-in-words	Contains the larger of either: a. size of initial reserve (in words), or b. highest track referenced (in words).

Word 9

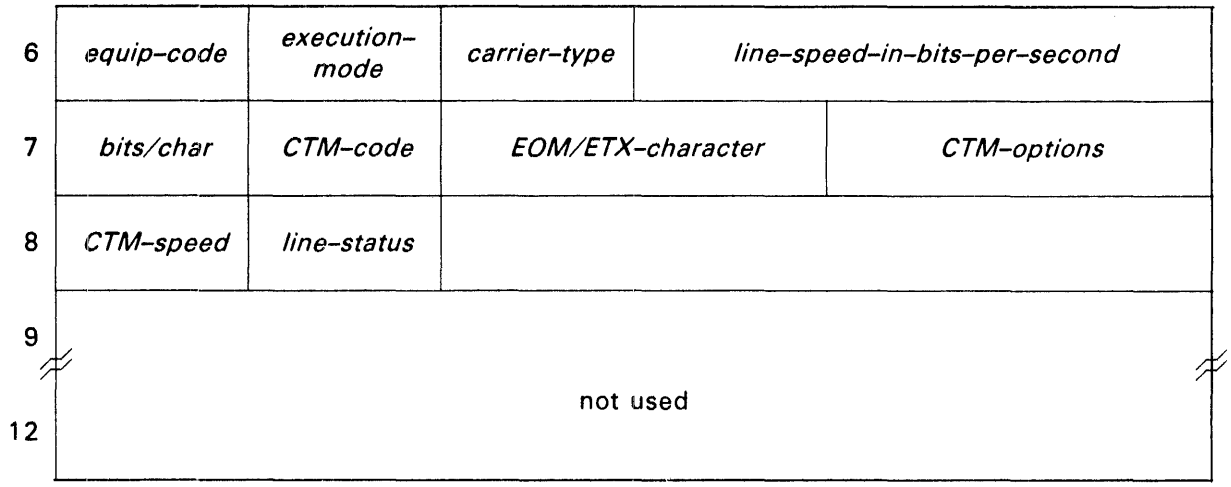
maximum-file-length	Contains the file maximum from the @ASG statement or the system standard maximum if that field is void.

Word 10

equip-code	equipment code of last facility upon which allocation occurred or of the facility containing the (cataloged) file's directory information or zero. (See Table 7-1.)
sub-code	sub-code of the above equipment type. (See Table 7-1.)

7.2.6.5. Communications Peripherals

The FITEM$ request packet format is:



Word 6

equip-code — (See Table 7-1.)

execution-mode — CTMC/GCS only:

0 - Half-word
1 - Quarter-word

carrier-type

0 - Leased line
1 - Telephone
2 - TELEX

Word 7

bits — Number of bits per character (including parity)

CTM-code — Indicates type of CTM:

0 - Standard (monitor)
1 - NASA
2 - GSA3EI (external)
4 - CTM IV (external interrupt with status word)
6 - CTM VI (external interrupt with status word)
7 - CTM VII (external interrupt with status word)
8 - CT ISO (external interrupt with status word)
9 - CT TRANSPARENT (external interrupt with status word)

CTM-options

Options are (indicated bit set):

Bit 11 - Hardware polling (CTA/CTS ISO)
10 - Full duplex
9 - Continuously emitting mode
8 - Release function indicator (for ring interrupt feature)
7 - EOM/ETX character passed as status word
6 - Even (0 bit) or odd (1 bit) parity indicator
5 - Block parity character transferred to CPU
4 - Space-to-mark transition
3 - Block parity, character parity, or late input acknowledge
2 - Ring indicator
1 - Carrier off
0 - Send data with idle character

Word 8

CTM-speed

Units speed. Codes are:

0-7 - Low
010-017 - Medium
020-037 - High
040-047 - TELPAK

line-status

Indicates status of line as follows:

Bit 20 0 - Line is not initialized for idle line monitor.
1 - Line is initialized for idle line monitor.

Bit 21 - Unused.

Bit 22 0 - Line is available.
1 - Line is not available.

Bit 23 0 - Line is operable.
1 - Line is down.

Bit 24 0 - Line (input, output, and dial) is not reserved.
1 - Line (input, output, and dial) is reserved.

Bit 25 0 - Line (input, output, and dial) is not assigned.
1 - Line (input, output, and dial) is assigned.

Table 7-1. Equipment Codes Returned by FITEM$

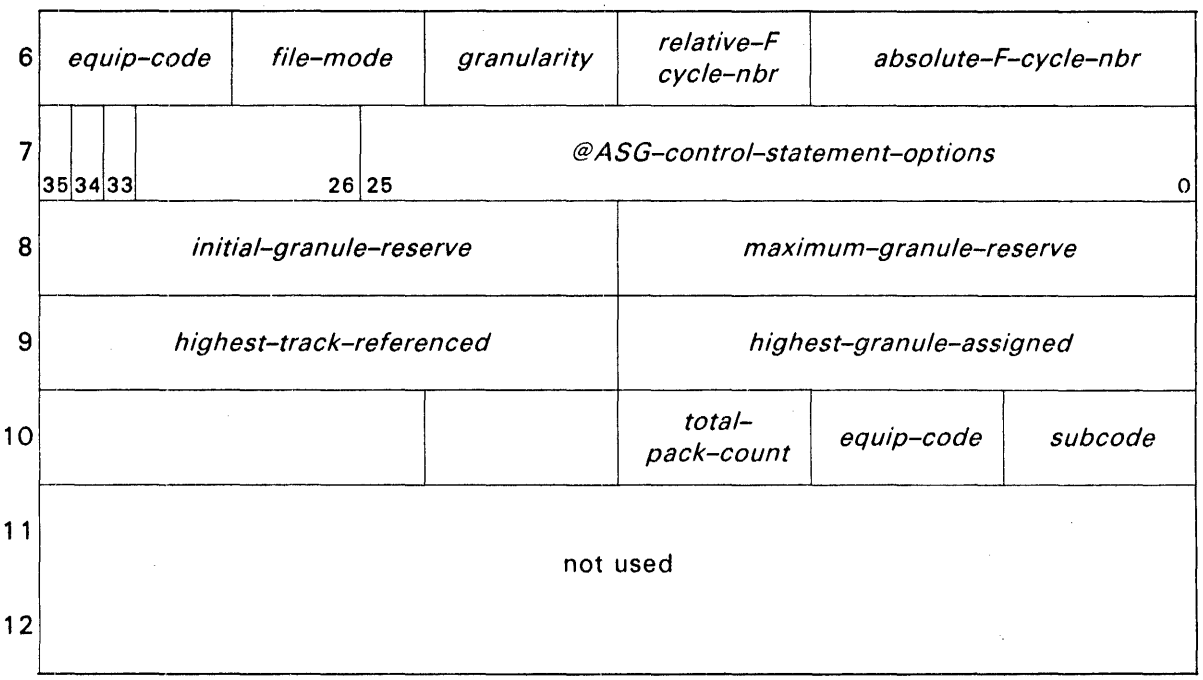
Equipment Code (Octal)	Equipment Type
0	In the case of a FACIL$ or similar packet, this can indicate a cataloged file in rollout status.
1	UNISERVO VIIIC Magnetic Tape (7-track)
2	UNISERVO VIC Magnetic Tape (7-track)
3	UNISERVO VIC Magnetic Tape (hardware translate 7-track)
4	UNISERVO VIC Magnetic Tape (hardware translate 7-track)
5	UNISERVO VIIIC Magnetic Tape (9-track)
6	UNISERVO VIC Magnetic Tape (9-track)
7,10	Not used
11	UNISERVO 12 Magnetic Tape (7-track)
12	UNISERVO 14/16/30 Magnetic Tape (7-track)
13	UNISERVO 12 Magnetic Tape (9-track)
14	UNISERVO 14/16 Magnetic Tape (9-track)
15	UNISERVO 20/30/32/34/36 Magnetic Tape (9-track)
16,17	Not used
20	FH-432 Magnetic Drum
21	Not used
22	FH-1782 Magnetic Drum
23	UNIVAC 8414, 8424, 8425 Disk (word-addressable)
24	UNIVAC 8440, 8430, 8433, 8434, 8405-00, 8405-04, 8450 Disk (word-addressable)
25	Unitized Channel Storage or Extended Storage
26,27	Not used
30	FASTRAND Drum II or III Magnetic Drum Subsystem or UNIVAC 8460 Disk

Table 7-1. Equipment Codes Returned by FITEM$ (continued)

Equipment Code (Octal)	Equipment Type
	Subcodes are used to differentiate among FASTRAND Drum II, III, and 8460 Disk equipment as follows: FASTRAND Drum II 00 FASTRAND Drum III 01 8460 Disk 02
31	Not used
32	Sector-formatted FH-432 Magnetic Drum
33	Not used
34	Sector-formatted FH-1782 Magnetic Drum
35	Sector-formatted 8414 ,8424 ,8425 Disk Subcodes are used to differentiate the 8414, 8424, and the 8425 disk equipment as follows: 8414 00 8424 01 8425 02
36	Sector-formatted 8440, 8430, 8433, 8434, 8450, 8405-00, 8405-04 Disk Subcode of zero is returned for all 36 code disks
37	Sector-formatted Unitized Channel Storage or Extended Storage
40 thru 56	Not used
57	Paper Peripherals
60	Console
61 thru 71	Not used
72	Communications Terminal Module Controller (CTMC) or General Communication Subsystem (GCS)
73	Communications/Symbiont Processor
74 thru 76	Not used
77	Arbitrary device

7.2.6.6. Removable Disk Files

The FITEM$ request packet format is:



Word 6

equip-code	Equipment code of the assigned facility. (See Table 7-1.)
file-mode	Bit 29 set – Exclusively assigned file Bit 28 set – Read key is needed Bit 27 set – Write key is needed Bit 26 set – Read-only file Bit 25 set – Write-only file
granularity	All zeros indicate track granularity; nonzero indicates position granularity

Word 7

bit 35	If set, system has tape label checking turned on.
bit 34	If set, file assigned as a temporary file.
bit 33	If set, internal name is a use name.
@ASG-control-statement-options	Indicates the options specified on the @ASG control statement (see 3.7.1) that assigned the equipment. Master bit notation is used; i.e., a 1 in bit 25 indicates the A option was specified, a 1 in bit 24 indicates the B option was specified, etc.

Word 9

highest-granule-number-assigned	The highest numbered granule (track or position) assigned to the file. This granule is the relative position of the granule within the file.
highest-track-referenced	The highest numbered track (track only) which has been written

Word 10

total-pack-count	Number of removable pack-ids.
equip-code	Equipment code of last facility upon which allocation occurred or of the facility containing the (cataloged) file's directory information or zero. (See Table 7-1.)
sub-code	Sub-code of the above equipment type. (See Table 7-1.)

7.2.7. Alternate Methods of Retrieving Facility Assignment Synopsis

These linkages (FACIL$ AND FACIT$) obtain a subset of the facility assignments previously defined in 7.2.6. To obtain the first nine words of the FITEM$ packet, the following linkage may be used:

```
L,U  A0,pktaddr
ER   FACIL$
```

To obtain the first 10 words of the FITEM$ packet, the following linkage is used.

```
L,U  A0,pktaddr
ER   FACIT$
```

It is not recommended that these linkages be used in any new programs.

7.2.8. Tape File Initialization (TINTL$)

Purpose:

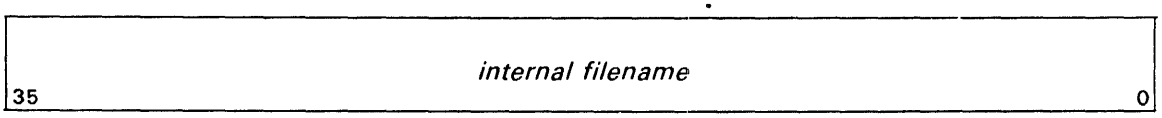
Causes a specified tape file to be logically rewound so that a subsequent pass can be made from the load point of the first reel.

Format:

```
L    A0,(,pktaddr)
ER   TINTL$
```

Description:

Pktaddr is the address of a 2-word request packet whose format is:



Access to the reel currently in use is closed. If the initial reel of the file is mounted on the first unit, the reel is simply rewound. Otherwise, the reels on the units involved are rewound, and a LOAD message is issued in order to have the initial reel mounted. If there are two units involved, they may be switched in order to ensure that the initial reel is mounted on the first unit.

7.2.9. Tape Swapping (TSWAP$)

Purpose:

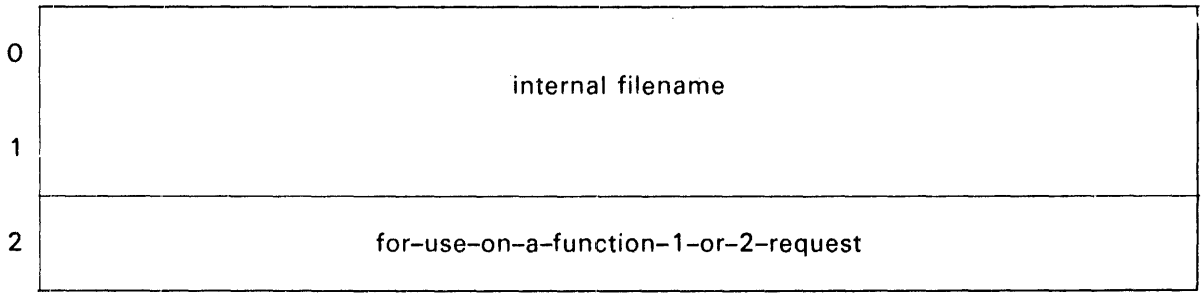
Closes access to the current reel of a tape file and requests loading of the next reel of the file (unless special action is requested).

Format:

```
L    A0,(function,pktaddr)
ER   TSWAP$
```

Description:

Pktaddr is the address of a 2- or 3-word packet whose format is:



Inclusion of a function in register A0 indicates the following:

Function	Description
0	Standard tape reel swap.
1	Standard tape reel swap occurs and the reel number of the tape swapped to is returned in word 2 (the third word) of the request packet.
2	A nonstandard request. A request is made to mount the reel specified in word 2 (the third word) of the request packet. If this reel is not currently recorded as part of the file, it is added as the last reel.

Access to the reel currently in use is closed and the reel is rewound. A request is issued to mount the next reel of the file. If two units are involved in the assignment, the LOAD request specifies the unit other than that which was just in use.

Some guidelines for the usage of ER TSWAP$ with two units are:

- On a two-servo, non-N option assignment of a tape file, there is a request to mount two reels, one for each servo.
- On a TSWAP$ request, a third reel is requested to be mounted on the first servo if more than two tapes were specified on the tape assignment.
- If two or fewer tapes were requested on the assignment, the operating system assumes that the assignment is a two-tape file and does not go to the third tape until it is requested (by the user). This logic is intended to support the frequent occurrence of files consisting of two tapes.
- The exception to the preceding occurs with the use of TSWAP functions 1 and 2. Whenever either of these functions is used, TSWAP's lookahead function is lost.

There is an advantage to using TSWAP$ with two units if a programmer wishes to have a 2-reel file, or if more than two tapes are used and all of the reel numbers are specified on the tape assignment.

7.3. TAPE LABELING

Tape labeling has two objectives: first, tape labeling is used to minimize the impact of operator and user error. This error can be either intentional or unintentional. Second, it provides a facility which allows a smooth data transfer between computing systems via magnetic tape. These computing systems may be logically close (may be under common managerial control) or they may be quite apart. In either case, a need exists to provide a functional interface between data on one system and data on a different system. In this instance, data interchange is the primary objective of a tape labeling facility.

One of the primary objectives of tape labeling is to provide a way that allows a specific site to define its position with respect to the extremes of data integrity/security and data interchange. The following describes a starting point, a "common ground" which is a tape labeling facility designed to fulfill most sites requirements. This "common ground" is also the basis for a facility that allows sites to define their own tape labeling facility.

7.3.1. Definitions

This subsection provides definitions of the terms used in connection with tape labeling and of the labels used by both the system and the user.

file section – That part of a file recorded on any one volume. The sections of a file may not have sections of other files interspersed.

header label group – A label group consisting of a system volume header label (if it is the first label group on the volume), system file header labels, and a tape mark. Optionally, under the control of the user, the group may also contain user file header labels following the appropriate system labels.

label – An 80-character block at the beginning or end of a volume or file serving to identify and/or delimit that volume or file. It is not considered part of the file section it delimits.

label group – A collection of contiguous labels pertaining to a file which precedes or follows that file or part of that file on a volume. The

volume header label(s) and the following file header label(s) are considered to be the first label group on the volume.

label identifier – A name consisting of three alphabetic characters identifying the type of label. It is followed by a label number.

label number – Any numeric digit 1 through 9. In label numbers, 0 (zero) is not used.

system labels – Those labels controlled by the operating system and which the user may not access directly. System end-of-volume and system end-of-file labels are mutually exclusive in any given end label group. The system labels are:

- VOL1 = Volume header label
- HDRn = File header label. Must be in consecutive ascending order and symmetric to EOF/EOV labels.
- EOFn = End-of-file label (corresponds to HDRn with the same rules applying)
- EOVn = End-of-volume label (corresponds to HDRn with the same rules applying)

tape mark – A special code configuration or sequence recorded on magnetic tape indicating the boundary between files and labels and also between certain label groups. The presence of the tape mark is generally detected by the tape unit or by the tape unit controller and is thereby signaled to the Executive.

user labels – Those labels whose reading, writing, and contents (except for the initial label identifier and number) are controlled by the user. They are:

- UHLa = User file header label. No order or symmetry of occurrence is necessary.
- UTL = User Trailer Label. Also known as user end-of-volume (corresponds to UVLn) or user end-of-file (no correspondence to UHLa is necessary) labels, depending on the circumstances.
- UVLn = User volume label. Corresponds to VOL1; use of n is optional.

volume – A physical unit of storage media. In this specification the medium is magnetic tape, so a volume is a reel of magnetic tape.

7.3.2. Structure of Magnetic Tape File

SINGLE FILE, SINGLE VOLUME

VOL1 HDR1 HDR2*—FILE A—*EOF1 EOF2**

SINGLE FILE, MULTIVOLUME

VOL1 HDR1 HDR2*—FIRST PART FILE A—*EOV1 EOV2**

VOL1 HDR1 HDR2*—LAST PART FILE A—*EOF1 EOF2**

MULTIFILE, SINGLE VOLUME

VOL1 HDR1 HDR2*—FILE A—*EOF1 EOF2* HDR1 HDR2*—FILE B *EOF1 EOF2**

MULTIFILE, MULTIVOLUME

VOL1 HDR1 HDR2*—FILE A—*EOF1 EOF2*HDR1 HDR2*—FIRST PART FILE B—*EOV1 EOV2**

VOL1 HDR1 HDR2*—CONTINUATION OF FILE B—*EOV1 EOV2**

VOL1 HDR1 HDR2*—LAST PART OF FILE B—*EOF1 EOF2*HDR1 HDR2* FILE C—*EOF1 EOF2**

NOTE:

** indicates a tape mark.*

7.3.3. Reading and Writing Tape Label Blocks (TLBL$)

Purpose:

Enables the user to read or write selected portions of HDR1, HDR2, EOF1, EOF2, EOV1, and EOV2. It also reads and writes user labels before and after each file. This ER is the user execution level facility referred to in the discussions of these labels.

Format:

```
L    A0, (buffer length, buffer address)
ER   TLBL$
```

Description:

Buffer length specifies to the Executive the length of the buffer supplied by the user. For read, write, and insert of system labels, this buffer can be any length needed. See Table 7-2. For read and write of user labels, the length must be at least 027 words. See Table 7-3.

Format:

Table 7-2. Read/Write System Label Packet

00			FCODE
01	internal filename		
02			
03	file identifier		
04			
05			
06			
07		record format	block length
010	record length		buffer offset length
011			expiration date
012	reserved		
013	reserved		
014	reserved		
015		accessibility code	reserved
016	block count		
017	reserved		
020			
021			
022		0	

The above format is used to read, write or insert system label information.

FCODE – Specifies the function to be performed. The allowable function codes are:

001 Read selected information from HDR1 and HDR2. This function is allowed only when positioned in the header label group or trailer label group of a tape file. If the user is positioned in a trailer label group, the header labels of the next file are read. The data read is placed in words 7 through 022 of the packet. FCODE and internal filename are required as input for this function; file identifier may optionally be specified.

002 Read selected information from EOF1 and EOF2 or from EOV1 and EOV2. This function is allowed only when positioned in the trailer label group of a tape file. The data read is placed in words 7 through 022 of the packet. FCODE, and internal filename are required as input for this function; file identifier may optionally be specified.

003 Read a UHL. This function is allowed only when positioned in the header label group or trailer label group of a tape file. If the user is positioned in a trailer label group, the header labels of the next file are read and label checked and then the UHLs of that file are read. UHLs are read as they are encountered in the header label group. The format of this packet is described in Table 7-3.

005 Read a UTL. This function is allowed only when positioned in the trailer group of a tape. UTLs are read as they are encountered. The format of this packet is described in Table 7-3.

020 Allows the user to specify selected information that is to be placed in the header label group of a tape file and it causes the HDR1 and HDR2 to be written at the time that the ER TLBL$ is issued. The selected information is specified in words 03 through 022 of the ER packet. This function is allowed only when positioned in the header label group or in the trailer label group. If the user is positioned in a trailer label group, the header label group of the next file is read and label checked using the data currently contained in the PCT tape item extension and then the header label group is written using the data in the packet. The data supplied by the user is retained in the PCT tape item extension and used to write the trailer label group for the file.

021 Write an 80-character UHL on the tape file. The format of the packet for this function is described in Table 7-3. This function is allowed only when positioned in the header label group or the trailer label group. If the user is positioned in a trailer label group, the header label group of the next file is read and label checked using the information in the PCT tape item extension. The HDR1 and HDR2 are written followed by the UHL. Multiple UHLs may be written by making successive ER calls. UHLs are written in the order in which they are received by the Executive.

022 Write an 80-character UTL on the tape file. The format of the packet for this function is described in Table 7-3. This function is allowed only when positioned in user data or in the trailer label group. If the tape is positioned in user data, EOF1 and EOF2

labels are written before the UTL is written. Multiple UTLs may be written by making successive ER calls. UTLs are written in the order in which they are received by the Executive.

023 This has the same function as 022 except that EOV1 and EOV2 labels are written if the ER is issued when the tape is positioned in user data.

060 Insert the information contained in the ER packet into the PCT tape item but do not write a label. This allows the user to change the information used to do a label check and the information used by the Executive to write the header label group. For example, this function could be used to insert the correct file identifier of the next file on tape into the PCT. Then a function 020 can be issued with a different file identifier in the packet and the file passes label check before the write is done. This function is allowed when positioned in the header label group and the headers have not been written or read, or when positioned in the trailer label group.

internal filename – The 12-character Fieldata internal filename to which the ER TLBL$ applies. This must be a tape file.

file identifier – 17 ASCII characters which are used as the new file identifier for write functions and the insert function. This field is used for label checking for read functions. If this field is set to zero, the value currently contained in the PCT tape item is used.

record format[1] – This field contains one ASCII character: F, D, S, or U. A value of zero in this field indicates that the record format is not to be changed. If the user specifies any other character, this cell in the PCT tape item is set to the system default value.

block length[1] – For the write and insert functions the maximum block length allowed is 99999. If the maximum is exceeded, 99999 is used. A value of minus zero in this field indicates that the block length should be set to zero. A value of zero indicates that the block length should not be changed.

record length[1] – For the write and insert functions the maximum record length allowed is 99999. If the maximum is exceeded, 99999 is used. A value of minus zero in this field indicates that the record length should be set to zero. A value of zero indicates that the record length should not be changed.

buffer offset length[1] – For the write and insert functions the maximum buffer offset length is 99. If the maximum is exceeded, 99 is used. A value of minus zero in this field indicates that the buffer offset length should be set to zero. A value of zero indicates that the buffer offset length should not be changed.

expiration date – For the write and insert functions, this field allows the user to specify a new expiration date. For files other than the first file, the expiration period must not exceed the expiration period of the previous file. If an expiration period greater than the previous one is specified, the previous file's expiration period is used. For read functions, this field has no meaning.

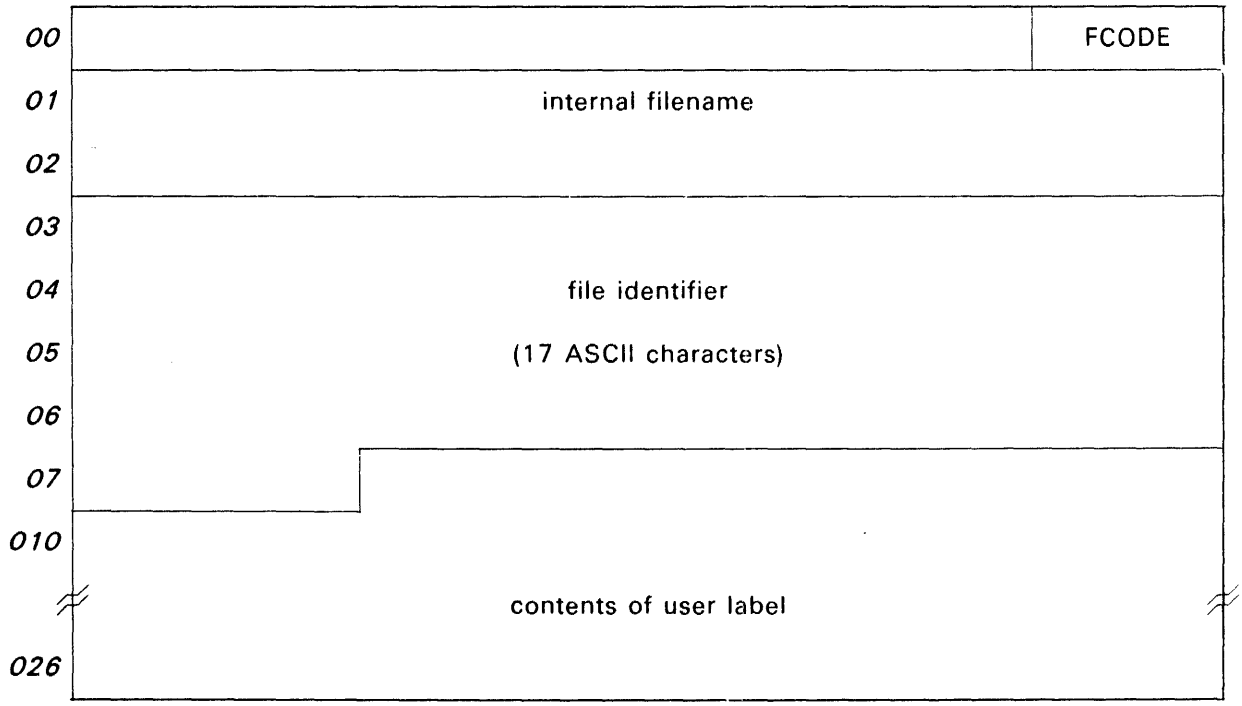
accessibility code – For the write and insert functions the allowable values are 061 through 067 and the ASCII space (040). If a zero is specified by the user, the current value in the PCT tape item is maintained. If an illegal value is specified, the options specified on the assign statement dictate what value is used for the new accessibility code. For read functions, the value returned is the accessibility code of the file at which the tape is currently positioned.

block count – For the write and insert functions this field has no meaning. When reading a header label group, this field is returned as zero. When reading a trailer label group, this field contains the number of user data blocks on the file to which the trailer label applies.

NOTE:

[1] The definitions of these fields are in the American National Standard for Magnetic Tape Interchange (ANSI X3.27-1969).

Table 7-3. Read/Write User Label Packet



The above packet is used to read or write a UTL or UHL.

FCODE – The function codes are defined above. The applicable functions for this packet are read a UHL (03), read a UTL (05), write a UHL (021), and write a UTL after an EOF (022), and write a UTL after an EOV (023).

internal filename – The 12-character Fieldata internal filename to which the ER TLBL$ applies. This must be a tape file.

file identifier – The file identifier is used only for the read functions and insert function. It is required if the file identifier currently in the PCT tape item does not match the file identifier of the file to which the read applies. If the file identifier is specified in the packet, it is transferred to the PCT tape item. If this field is set to zero, the file identifier currently contained in the PCT tape item is used. For the write functions, this field contains the first 17 characters of the user label to be written.

user label – For write functions, words 3 through 026 contain an 80-character ASCII user label. The first three characters must be either an ASCII UHL or UTL to identify the label as either a user header label or user trailer label. The fourth ASCII character must be from 1 through 9. The remaining 76 ASCII characters are for user application. For the read functions, a user label is placed in words 3 through 026 of the ER packet. The packet must always be large enough to contain 80 ASCII characters starting at word 3.

■ ER TLBL$ Errors

On return from the ER TLBL$, A0 is set positive to signify normal completion, and is negative if the operation was unsuccessful. If negative, S3 of A0 contains an error code. See Table 7-4 for error codes and their meanings.

Table 7-4. Error Codes and their Meanings

Code	Meaning
001	The user ER packet is wholly or partially outside of the program's limits.
002	A reference was made to an unassigned file.
003	File is being freed.
004	Attempt to process an insert function when the tape is positioned beyond the header label group or in user data.
005	Read or write of a label was attempted after an end of volume was encountered.
006	The function specified in the ER TLBL$ packet is illegal when positioned in user data.
007	An undefined function code was specified.
010	Attempt to read a user label following a write of a user label.
011	An I/O error was encountered. The I/O error code is in S2 of A0.
012	ER TLBL$ to a labeled tape which was created by an earlier version of tape labeling (9 bit label).
013	Read or write in the trailer label group was attempted while positioned in the header label group.

Table 7-4. Error Codes and their Meanings (continued)

Code	Meaning
014	The user buffer is too small.
015	Failed label check
016	ER TLBL$ to an unlabeled tape or tape assigned with low density specified
017	Physical load point encountered on a move backward request issued by the Executive. This error is encountered when doing reverse processing.
020	Attempt to write a UTL without writing or rewriting the EOF or EOV labels.
021	The label requested by the ER TLBL$ read function could not be found.
022	A syntax error was encountered in the first four characters of a user label.
023	The file specified in the packet is not a tape file.
024	Tape positioned in EOV label while processing HDR1, HDR2, EOF1 or EOF2. This is a warning only. A0 is not set negative.

7.3.4. Reading Tape Label Blocks (LABEL$)

Purpose:

Enables the user to read any 9-bit ASCII character (old format) label block in the first label group on the volume except the VOL1 block.

Format:

```
L,U  A0,pktaddr

ER   LABEL$
```

Description:

Pktaddr is the address of the packet whose format is:

00	ASCII-Fieldata translation	read label block	unused	label buffer address
01	internal filename			
02				

Word 1

ASCII–Fieldata translation — If equal to 1, indicates ASCII format label block.

If equal to 0, the read packet is to be translated to Fieldata format.

read label block — If equal to 0, read a label block.

label buffer address — Address of a 20–word label buffer is required for read label requests.

Control is returned to the user at the location immediately following the LABEL$ request after the request has been completely processed. Register A0 contains the following:

S1 If set to 0, indicates normal completion; if set to 40_8, indicates abnormal completion (check S2 and S3).

S2 Contains I/O status (if 0, all I/O has completed normally; if a nonzero value, see Appendix C for abnormal I/O status.

S3 If set to 1, indicates invalid label buffer address.

If set to 4, indicates invalid filename.

If set to 10_8, indicates an invalid request (not 0).

If set to 12_8, indicates a request on a nonlabeled tape.

If set to 13_8, indicates a request on an 8–bit ASCII character (new format) labeled tape.

If set to 14_8, indicates the request failed label verification.

H2 The label buffer address originally supplied by the user

7.3.5. Multivolume Processing

If an ER TSWAP$ is requested on output, the tape file is closed with

```
*EOV1  EOV2**
```

If the last I/O function to the tape was a write tape mark, a void file is created before closing the tape:

```
*HDR1  HDR2**EOV1 EOV2**
```

The information used to create the label is the last information received by the system to write or verify labels. Therefore, if new information is required in the end of volume labels or if user trailer labels are desired, the ER TLBL$ requests must be made prior to the ER TSWAP$ request.

7.3.6. Reverse Processing

The reverse processing of label groups is handled the same as forward processing. The direction of processing only determines which labels to check after a tape mark is encountered or which labels to check before the user's next I/O.

7.4. DISK LABELING

A label record is written on a disk pack when it is prepared by the Executive (see 6.7) for use as mass storage. This record is only accessible to and is used exclusively by the Executive. It contains the pack identity and all the information necessary to establish Executive control over the pack. The pack-id field of the label record is used at assign to guarantee that the requested pack is received (see 3.7.1).

8. Demand Processing

8.1. INTRODUCTION

Demand processing is a mode of operation in which run processing is dependent on manual interface with the Executive during processing. Basically, it is a conversational mode of operation requiring a demand and response type of activity. Conversational operation via a remote terminal causes the Executive or an active program to immediately react and respond. Demand processing terminals are generally remote from the computer site. They usually have a printer or a cathode ray tube and a keyboard. An example of a demand terminal is the teletypewriter keyboard and printer.

The distinction between batch-mode processing and demand processing lies in the frequent interaction with the user that occurs during demand processing. The terminal user is considered to be in conversation with the Executive, a special demand function, user programs, or the batch functions of the Executive on a task basis.

Tasks executed by the demand terminal user normally have frequent but short bursts of computation. Progress is always insisted upon; however, to receive a substantial amount of computation may require a long period of time. Access to computation is a percentage of the total computing facility and is scheduled in small increments of time at frequent intervals to provide immediate responses. This action gives the appearance of total system control to the user and the impression that he is the only user currently running. The more a user is required to interact with a demand program, the shorter the bursts of computation required to service a given request. The bursts of computation are time-shared within the Executive to provide an apparent immediate response, with the program placed in a dormant mode during idle periods awaiting response from the user.

While a demand program is in a dormant mode, it may be necessary to swap the program from main storage. Normally, this happens only when main storage is full and another program, which is currently on mass storage, has work to do.

The Executive supports the use of the following terminals to access the system in the demand mode:

- UNISCOPE 100/200 Display Terminal
- Universal Terminal System 400 (UTS 400)
- UNIVAC DCT 1000 Data Communications Terminal
- UNIVAC DCT 524/500/475 Data Communications Terminal (Teletypewriter)

- Teletype Models 33, 35, 37, 38*, 38A*, and (KSR and ASR)
- Tektronix 4013*

* Teletype Models 38 and 38A and Tektronix Model 4013 are supported only when connected via a CTMC or GCS.

8.1.1. General Demand Terminal Operational Procedures

The following is a brief discussion of the operational procedures for initialization of a demand terminal, submission of a demand run, termination of the demand run, and deactivation of the demand terminal.

8.1.1.1. Initialization

Before the demand terminal can be initialized, the user must turn it on, set the various switches to the proper position, and establish the proper line connection if operation is on a switched line network. A discussion of the hardware characteristics of the various terminals can be found in the appropriate programmer/operator reference manuals. The manuals describing the operational procedures for SPERRY UNIVAC equipment are:

- SPERRY UNIVAC UNISCOPE Display Terminal, Operator Reference, UP-7788 (current version)
- UNIVAC DCT 500 Data Communications Terminal, Operator Reference, UP-7832 (current version)
- UNIVAC DCT 1000 Data Communications Terminal, Operator Reference, UP-7827 (current version)
- SPERRY UNIVAC Universal Terminal System 400, Operator's Guide, UP-8358 (current version)

Once the connection is made and the terminal is initialized, the demand user must send a 6-character remote site-id to the operating system. Each terminal is assigned a site-id, and the Executive maintains a list of all valid site-ids. The site-id submitted by the demand user is compared to this list. If the system responds with the message

ENTER USERID/PASSWORD

the demand user can assume that the initialization operation is completed If the site-id is invalid or already in use, the site attempting to establish communications is not recognized. If the message is not received within approximately one minute, the demand user should retransmit the site-id (the previous transmission may have been received in error by the Executive).

NOTE:

If the Terminal Security System is not configured, the initialization operation differs from the above procedure (see 8.3).

8.1.1.2. Demand Terminal Modes of Operation

The demand terminal has three distinct modes of operation. They are described below.

- Terminal Inactive Mode

  The initial mode of the terminal following the sign-on procedure. The terminal returns to this mode at the completion of the other two modes.

- Demand Run Mode

  This mode is achieved by submitting an @RUN control statement (see 3.4.1) from the primary input device; that is, the keyboard. The terminal operator must wait until the date and time message is displayed at the terminal before submitting the runstream data. In demand mode, the input is solicited when input is desired by the Executive. The terminal is returned to the inactive mode by submitting an @FIN control statement. Another @RUN control statement is not accepted while in the demand run mode. This operation will usually be done automatically for the user by the Terminal Security System.

- Remote Batch Mode

  The demand terminal may be switched from the terminal inactive mode to the remote batch mode for input or output.

  The "B" operation on the @RUN control statement (@RUN,/B) places the terminal in the remote batch input mode. Input is not solicited as in demand run mode. Output is not displayed at the terminal while in this mode. The terminal is returned to the terminal inactive mode following an @FIN control statement. Another @RUN control statement is accepted while in the remote batch input mode and is treated as another remote batch run whether it contains a B option or not. Output files generated by the remote batch run as well as those SYMed (via @SYM) to the terminal can be displayed at the terminal by entering the statement, @@SEND (see 8.1.1.3.1). This mode can be entered only from the terminal inactive mode. The terminal is returned to the terminal inactive mode when the output process of the file is complete.

8.1.1.3. Demand Symbiont Interface

All Sperry Univac-supported demand terminals use a common symbiont interface for input and output processing. This provides several controls and features to all demand terminals in a uniform manner.

8.1.1.3.1. Demand Symbiont Control Statements

Control of demand symbionts is regulated by control statements prefixed with a double master space or at symbol (@@).

These control statements do not require the input solicitation. They may be entered from the terminal after an output interrupt (break-key) or at any other time the terminal operator finds the need. They may also be presented from the user program via the Executive Request PRTCN$ (see 5.4). When PRTCN$ is used, it should be noted that the image is processed serially with the other output. The control statements are listed in Table 8-1.

Table 8-1. Demand Symbiont Interface Control Statements

Statement	Mode	Description
@@CM *text*	All	The @@CM causes text to be displayed at the operator's console. This control statement may be entered outside run mode.
@@CONS	All	Causes all input not prefixed by @ to be taken as an operator keyin.
@@CONT	All	Directs the symbiont to continue. Useful after a BRK-KEY when no action is desired.
@@CQUE	Demand Run	Inhibit input solicitation. Allow several input images to be buffered in memory before the terminal is placed in the wait condition. Allows the user to continue typing input while previous input is being processed.
@@END *type*	Demand	Terminates special modes; i.e., @@CQUE, @@INQ, or @@ESC. The @@END with no type given returns the terminal to solicit mode from @@CQUE and @@ESC and begins processing the mass storage buffered input for @@INQ. *type* may be: ESC - Escape mode INQ - Terminate INQ mode CONS - Console mode CQUE - CQUE mode FUL - Full screen mode PTI - Paper Tape input HOLD/PR/TM - Clear hold conditions as specified.
@@ESC [I] [O]	Demand Run (UNISCOPE 100/200 Display Terminal)	The I option allows input to be passed to the requestor unaltered from the format of which it was entered; that is, all communications envelope characters are not removed nor is the image translated. The image consists of the characters following the STX, up to and including the ETX. The O option allows the program to prepare similar output.
@@HOLD *type / type*	Demand Run	Place a hold on type specified as either "TM" for terminal message and/or "PR" for print output. If *type* is not specified it defaults to "PR" and "TM". Any system generated terminal messages are held when "TM" is specified. This includes output individually formatted to respond to an @@ command; e.g., -@@ERROR xxxxx. System canned messages, such as -@@COMPLETE, will continue to be displayed regardless of HOLDs specified.

Table 8–1. Demand Symbiont Interface Control Statements (continued)

Statement	Mode	Description
@@INQ	All	Directs the Executive to buffer all input to mass storage until the @@END control statement is received. If the @@INQ statement is entered in terminal inactive mode, the next input should, of course, be a @RUN statement. All run statements entered while in @@INQ mode are considered remote batch and not demand.
@@RQUE	Remote Batch	Stop the present batch output file and requeue it for a later @@SEND request. Return to terminal inactive mode.
@@SEND,option [name[n]]	Terminal Inactive Demand Run	Sends queued batch output file to the terminal. C option specifies punch file, otherwise print file is assumed. U option specifies that files queued to the user–id active at the terminal are to be sent. No U option specifies that files queued to the demand terminal's site–id are to be sent. "Name" is either a specific filename or the run–id to be used in sending a PRINT$ or PUNCH$ file. If "name" is a run–id, "n" indicates the specific part number of the PRINT$ or PUNCH$ file. If "n" is absent, all parts are assumed. If only the @@SEND and options are specified, all files queued to either the active user–id or terminal site–id are sent, depending on the presence or absence of the U option.
@@SKIP n	Demand Run	Skip n lines of output where n is a value of 0 to 63. The SKIP may be reset by an @@SKIP 0.
@@TERM	All	Directs the Executive to terminate the terminal. If entered while a run is active, the run and terminal are terminated immediately.
@@TM *addressee text*	Demand Run	Sends text to the specified addressee. The addressee is specified as follows: run–id/R – Sends text to specified run–id (/R must be used as shown) user–id/U – Sends text to specified user–id (/U must be used as shown) site–id – Sends text to specified site–id The addressee must be active.
@@X TIOC	Demand Run	The @@X statement directs the Executive to take action on any or all of the four possible action parameters: T – terminate the demand run's present execution. I – discard all backed–up input. This includes all images from ADD files and CQUE mode. Input entered via @@INQ (includes FUL, PTI, TCI) will

Table 8-1. Demand Symbiont Interface Control Statements (continued)

Statement	Mode	Description
		be discarded only if the corresponding @@END was received prior to @@X I. O - discard all backed-up output. C - generate a Remote Break (RBK) contingency. If the user program has not registered to process the RBK contingency (see 4.9), action reverts to the "T" parameter if it is also present on the @@X request.
	Remote Batch	O - discard the remainder of the present batch output file. *NOTE:* *@@X with no parameters defaults to @@X TIOC.*

8.1.1.3.2. Transparent Control Statements

The demand symbiont control statements should not be confused with the transparent control statement, (3.2.8). Transparent control statements must follow certain demand symbiont rules of operation. These rules are:

1. Transparent control statements may be entered only in demand run mode.

2. They may not be entered from any means other than the primary input device, (i.e., at the keyboard and not paper tape, full screen, or cards).

3. Processing of any previous transparent control statement from the same terminal must have been completed.

8.1.1.3.3. Demand Terminal Messages

The demand symbiont interface provides common interface messages for all types of terminals. The messages are self explanatory; however, the most important ones are listed below.

* UNIVAC 1100 OPERATING SYSTEM VERSION xx.xx.xx * (RSI)

This message announces recognition of the site which is now in terminal inactive mode.

* TERMINAL INACTIVE *

The terminal is now in an inactive mode such that the next Executive control statement expected is an @RUN (or @RUN,/B) or a RSI-CCR control statement.

* TERMINAL INACTIVE - OUTPUT FILE AVAILABLE *

The terminal is in terminal inactive mode and a batch output file is available. An @@SEND causes the file to be transmitted.

* NO RUN ACTIVE *

An input image was received other than an @RUN or a demand symbiont control statement while in terminal inactive mode.

* WAIT - LAST INPUT IGNORED *

The last input image received was rejected because:

1. In solicit mode without a solicit for input.
2. Another statement is being processed.

* RUN ALREADY ACTIVE *

An @RUN statement was received while in demand run mode.

* BATCH RUN MODE - ENTER RUNSTREAM *

An @RUN,/B has been received and this message announces that input is now desired.

* DEV NOT CONFIGD FOR OUTPUT FILES *

An @@ SEND was received from a demand terminal which is not configured for receiving batch output files.

* EXECUTION TERMINATED *

The execution of the demand run has been terminated by the demand symbiont @@X T control statement.

* SYSTEM HOLD ON DEMAND RUNS *

The onsite operator has placed a hold on the opening of demand runs. Demand symbiont control statements are accepted and the remote batch input mode (RUN,/B) may be entered to place runs in the backlog.

* SYSTEM WARNING - MAX TIME *

The estimated time on the @RUN statement has been exceeded. It is doubled and the program/run continues normally.

* SYSTEM WARNING - MAX PAGES *

The page estimate on the @RUN statement has been exceeded. It is doubled and the program/run continues.

* SYSTEM WARNING - MAX CARDS *

The card estimate on the @RUN statement has been exceeded. It is doubled and the program/run continues.

*TERMINATED DUE TO SYSTEM HOLD ON TERMINALS *

The onsite operator has placed a hold on the recognition of demand sites.

* TERMINATED DUE TO UNAVAILABLE SYSTEM RESOURCES *

The recognition of the site is not allowed at this time due to a lack of system resources and/or maximum number of terminals allowed are already active.

* MORE INFO? * (A, X, R, J, E)

This message is output when a demand program errors or aborts after an error identification line has been printed. This facility allows the demand user to obtain more information about the error. If no additional information is desired, another control statement may be entered. The responses available to this message are any combination of the following letters:

A - Printout of the A-registers
X - Printout of the X-registers
R - Printout of the R-registers
J - Jump stack saved at the time of the error (1110, 1100/40, 1100/80 only)
E - Message explaining the error code

MSG QUEUE FULL

The addressee on an @@TM already has 7 messages queued.

TM KEY ERROR

Syntax Error On @@TM

USER-ID NOT ACTIVE

The user-id on an @@TM is not currently active.

RUN-ID NOT ACTIVE

The run-id on an @@TM is not currently active.

SITE-ID NOT ACTIVE

The site-id on an @@TM is not currently active.

HOLD ERR - SYNTAX

Syntax error on @@HOLD

HOLD ERR - NOT DEMAND

@@HOLD is only allowed for demand runs.

END ERROR

@@END is not allowed for this command type.

@@ERR – MAX INPUT LENGTH 200 CHAR

User has tried to set input length greater than 200 characters on a TTY type device. 200 characters is the maximum allowed.

WARNING CONSOLE OUTPUT LOST

Some console output may be lost.

8.1.1.4. Demand Terminal Termination

The standard termination procedure is performed when an @FIN control statement is received by the system. The Executive retains control of the terminal line until all output destined for the site has been processed.

The symbiont then returns to the terminal inactive mode. Either another logon sequence or the termination sequence, @@TERM, should follow. The telephone connection is available for further communications until the termination sequence is transmitted from the terminal. The termination sequence leaves the telephone connection available for dedicated lines. A disconnect function is sent to hang up dial connections.

When the demand run is terminated with the termination sequence or by the onsite operator, the terminal is immediately released. The run is terminated with no indication of termination being sent to the terminal. Any information previously received by the symbiont and not processed is discarded. Likewise, any accumulation of output by the symbiont is also lost. If the console operator downs the subsystem and unit for a terminal, the symbiont treats the site as though it had terminated.

8.2. DEMAND SYMBIONTS

8.2.1. Teletypewriter/DCT 524/500/475 Symbiont

The TTY/DCT 524/500/475 symbiont provides support for Teletype Models 33, 35, 37, and 38 (KSR/ASR), Tektronix 4013, and the DCT 524/500/475 operating in teletypewriter mode. See 8.2.1.6 for initialization of the DCT 500/475 in teletypewriter mode.

8.2.1.1. Operational Considerations

Initialization, runstream submission, termination, and remote system messages are described in 8.1.1 with the following exceptions:

1. The symbiont allows paper tape input and output on all devices (8.2.1.2).

2. Three characters are recognized as having special significance (see 8.2.1.3).

3. If a *PARITY ERROR* message is displayed at the terminal, the symbiont has detected a parity error on a transmission from the terminal. The transmission is ignored.

4. The symbiont recognizes a variety of parameters which the user may change to determine certain operation characteristics of the terminal (see 8.2.1.5).

8.2.1.2. Paper Tape Operations

Paper tape input/output is supported in an identical manner on ASR Teletypes and DCT 524/500/475's (either mode) as far as is possible.

8.2.1.2.1. Paper Tape Output Operations

No special effort is required to force output to paper tape on TTY mode devices. All that must be done is to turn on the paper tape punch, and output occurs on both the printer and the punch.

If the paper tape punch cannot be accessed due to some hardware problem, the following message is displayed:

PAPER TAPE PUNCH COULD NOT BE SELECTED

8.2.1.2.2. Paper Tape Input

Paper tape input is supported for tapes having the format data –CR–LF –data –CR–LF. The LF is needed only for readability at the terminal and may be deleted. The following procedures govern the use of paper tape input:

1. The paper tape reader should be prepared for input.

2. The user must enter the statement:

   @@PTI

3. The system responds with the message:

   START PAPER TAPE INPUT

4. The paper tape is read.

5. To terminate paper tape input, enter CNTRL–S (X–OFF or DC3) from the keyboard or from the tape (the latter is desirable).

   When paper tape input is completed, the symbiont responds with this message:

   END PAPER TAPE INPUT

   The @@END PTI statement must then be entered to terminate the paper tape input mode.

6. Paper tape input is terminated without operator intervention if a parity error is discovered or if data loss occurs.

7. If a real-time program prevents the symbiont from properly servicing paper tape input and data is lost, the following message is displayed:

   PAPER TAPE INPUT DATA LOST

The following are special considerations for the utilization of paper tape input:

1. A complete runstream may be entered on paper tape. The Executive treats this as a batch run. This must be done when the terminal is in inactive mode (see 8.1.1.2).

2. An @RUN without an @FIN may be entered via paper tape. The run is treated as batch input. Subsequent input from the keyboard is treated as a continuation of the batch input.

3. Paper tape input may contain nongraphic characters (which are normally illegal) if the terminal is in input escape mode (see Table 8-1).

8.2.1.3. Special Characters

The TTY/DCT 524/500/475 symbiont accepts several ASCII input characters as having special meaning. The interpretation of these characters may be changed dynamically by the user (see 8.2.1.5).

ASCII Character	Keyboard Position	Function	Description
CAN	CNTRL-X	Line Delete	The current image is discarded. The symbiont responds with a CR/LF sequence.
SUB	CNTRL-Z	Character Delete	One preceding character is deleted each time the SUB character is sent.
CR	RETURN or CR	End of Image	Used to indicate the end of an input image.

The output character > is used to denote input solicitation; i.e., the user may enter input.

8.2.1.4. Interrupting Output Processing

The break key is represented on the keyboard as BREAK or RTS or INT or INTERRUPT. It is used to temporarily terminate output so that the user may enter a demand symbiont control statement or a transparent (system) control statement.

Upon receiving the break/interrupt, the symbiont prints the following message:

OUTPUT INTERRUPT

8.2.1.5. Operation Modification Control Statements (@@TTY,@@DCT)

The TTY/DCT 524/500/475 symbiont provides the @@TTY and @@DCT symbiont control statements which allow the user to change certain parameters associated with the terminal. The @@TTY and @@DCT control statements are interchangeable (i.e., may be submitted from either device), and all parameters on the control statement are optional.

Format:

@@TTY char-1, value-1, ..., char-n, value-n

or

@@DCT

The character is a single alpha character selected from:

C - Character delete
I - Input image length
L - Line delete
P - Page length
S - Solicit
T - Image terminator
W- Page width

The value may either be a character (with the exception of page width, page length, or input image length) or an octal or decimal value. If a single digit is used, this is treated as a character.

Description:

As an example, to alter page width to 96, the image terminator to backspace and the solicit character to a colon (:), the following control statement suffices:

@@TTY W,96,T,010,S,:

For obvious reasons, the user may not define the termination character as T, line delete as L, or character delete as C.

8.2.1.6. DCT 524/500/475 in Teletypewriter Mode

The general information necessary to operate a DCT 500/475 in teletypewriter mode is given below. This paragraph deals with those changes involved in initializing a terminal.

- The DCT 524/500/475 hardware must be configured to appear to the system as if it were a teletypewriter. Specifically, the DCT 500 hardware must have the following:

  1. The RID, SID, and STX feature must be inhibited.

  2. The DCT 500/475 must be in the master mode.

  3. The DCT 500/475 full/half-duplex option must be set to the half-duplex mode.

  Once the terminal has established a line connection with the central site, the terminal operator must depress the PROCEED key to establish clear-to-send at the DCT 500/475. The CLEAR TO SEND indicator lights if the data set is in data mode when the PROCEED key is pressed. Once this sequence is performed, the terminal operator can send a site-id to the system.

8.2.1.7. Tektronix 4013

The general information necessary to operate a Tektronix 4013 is given in preceding paragraphs. This paragraph details those differences involved for the 4013.

1. To initialize, switches must be set as follows:

   - LOCAL/LINE must be set to LINE.

   - APL/ASCII–APL must be set to APL/ASCII.

2. The Executive performs no screen control for the user other than line spacing. The user must press RESET PAGE as needed.

3. The user program may place the terminal in APL mode via the ESC SO sequence. This mode is enforced until another line of output is sent to the terminal. The user may do this manually by typing ESC, CNTRL–N. To manually return the terminal to ASCII mode, enter ESC, CNTRL–O.

4. When the terminal has been placed in APL mode by the user program, solicitation occurs via the APL semicolon (;) unless changed by the @@TTY or @@DCT keyin. Normal solicitation occurs with the greater–than character (>).

5. See Tektronix 4013 manual for an explanation of other features.

8.2.2. UNISCOPE 100/200/DCT 1000 Symbiont

The UNISCOPE Display Terminal 100/UNISCOPE Display Terminal 200 (U100/U200)/DCT 1000 symbiont, in conjunction with the general demand symbiont interface, provides support for the following configurations:

- Combination of single, multiplexed, and multidropped UNISCOPE 100/200 Display Terminals (U100/U200) and associated Communications Output Printers (COPs) and Series 600 Tape Cassette Systems (TCS)

- DCT 1000, keyboard, printer, card reader, card punch, paper tape reader, and paper tape punch

- Wherever U100/U200 is indicated in this section, identical support is also provided for the UTS 400 Phase I terminal

U100/U200, UTS 400, and DCT 1000 Terminals can be mixed on a multiplexer or on multidropped lines.

8.2.2.1. Operational Considerations for UNISCOPE 100/200 Display Terminals

Initialization, runstream submission, termination, and remote system messages are as described in 8.1.1 with the following considerations:

1. Before the UNISCOPE 100/200 Display Terminal becomes active, the operator must turn it on and establish the proper line connection. After the device has been turned on and the line connection is made, the device is polled at intervals specified in system generation. The first message transmitted from an inactive device must be the 6–character site–id for this device. The site–id may be transmitted from anywhere on the screen, preceded by an SOE (▷) character.

If the transmitted message is received properly, the initialization message is sent to the device and the SOE and cursor character are positioned at the standard insert point for the next operator input. The SOE is intended as a solicit character, and indicates that the system is ready for input. If the message is not received properly, no response is sent to the device. If the operator is sure that everything is functioning properly and that the transmitted site-id is correct, then the following steps should be taken after a waiting period of about 60 seconds:

a. Press the wait switch (to unlock the keyboard).

b. Transmit the message again.

The operator sees a positive action when the device is polled to pick up the transmitted message by the reappearance of the cursor character. The cursor disappears from the screen when the transmit key is pressed and reappears when the device is polled.

2. Input sent to the computer consists of the data between the cursor and the previous SOE or between the cursor and the screen origin, if no SOE precedes the cursor. This input is broken into images, each image occupying at most one line of the screen. Unless full-screen input has been enabled (see 8.2.2.3), multiple images are not accepted. All are rejected if more than one is sent.

   After the operator has pressed the transmit key, the cursor disappears from the screen. If the message is received properly by the computer, any output is sent to the insert point, the symbiont generates and sends a keyboard unlock message, moves the screen up or down (depending on the current setting, see 8.2.2.3) one line, and positions an SOE character and the cursor in columns one and two of the insert line. The operator is free to enter a new line to be transmitted when the SOE appears.

   All data transmitted by the remote operator must follow submission of an @RUN control statement, except for special control sequences (see 8.2.2.3).

3. Operator input activity is timed to determine and prevent unnecessary use of communications facilities in intervals specified in system generation. The following symbiont action takes place at the successive intervals of no activity:

   1st Interval – A timeout message is sent.

   2nd Interval – Another timeout message is sent and the site is terminated.

4. No special action is required of the operator in handling output to the device. However, controls are available to perform functions such as skipping output lines, directing output to a printer, changing the insert point, etc. (see 8.2.2.3).

   The symbiont controls the screen with output messages.

   The operator may interrupt output via the MESSAGE WAITING key. The computer acknowledges the interruption, and the operator is then free to enter any input, specifically including demand symbiont control statements (see 8.2.2.3).

5. Except at initialization, the terminal operator need not be concerned with transmission error recovery. The demand symbiont complex controls all retry sequences. No retry sequences are performed on the initial activation message. If the retry sequences are unsuccessful, any active runs are terminated, all facilities are released, and the device is returned to the inactive state. The terminal operator must perform the initialization procedure after the transmission problem has been corrected. Any uncompleted terminal activity must be redone after the device is reinitialized.

6. It is the terminal operator's responsibility to remove the device from the active status when finished. This is done by transmitting the demand symbiont control message @@TERM. If the operator does not, the terminal times out.

■ Operational Considerations for the Series 600 TCS

Symbiont control messages for selection, deselection, track positioning, input and output are discussed in 8.2.2.3. Prior to using any of these symbiont control messages, the terminal to which the TCS is attached must be properly signed on as described in 8.2.2.1.

After seating the cassettes properly on the transports, the switch is pressed as follows:

1. POWER — Enable power to the Series 600 TCS. Be sure to also enable power to the COP or QTP if one is present.

2. CASS 1 and CASS 2 — Select either transport one or transport two for future operations.

3. REWIND — Rewind the selected transport to the Beginning of Tape (BOT). Each cassette seated on a transport should be rewound prior to other online or offline operations to align the track position counter with the BOT and assure the cassette is seated properly.

4. WRITE — Write select the TCS transport. If the transport is positioned at BOT, the transport moves to write load point. For offline operations, data in the U100/U200 memory between the cursor and the preceding SOE can be written as a block to tape by keying the print function.

5. READ — Read select the TCS transport. If the transport is positioned at BOT, the transport moves to read load point. For offline read operation, one block of data is inputted to the U100/U200 memory each time the print function is keyed in on the U100/U200 keyboard.

6. AUTO TR — When automatic transmit is on, a transmit function follows each tape cassette read. This switch must be set when reading tapes online with symbiont control messages.

7. < 1 BLK — Each time this switch is depressed, the selected transport positions the tape to the preceding Inter-Block Gap (IBG). This function ends with the transport read function selected.

8. SEARCH — Various search modes are available to position the tape on the selected transport with offline operations.

9. EDIT — The EDIT offline function allows tape cassette 1 to be edited to tape cassette 2. The edit operation is controlled from the U100/U200 keyboard.

10. LIST — The LIST offline function provides the capability to list a tape cassette on a QTP or copy a tape.

11. STOP — By pressing the stop switch, the selected transport for read or write operations is deselected. When reading or writing online, symbiont control messages should be used for deselection because a reselection sequence may be sent following the STOP switch operation.

For detailed descriptions of offline operations, reference UNIVAC Model 610 Tape Cassette System Component Description, UP-8012 (current version).

Except for the POWER switch, the only switch setting necessary for online operations is the AUTO TR switch when inputting data.

Inputting (@@TCI) and outputting (@@TCO) may be either in single image mode or multi-image mode operation. When the cassette is selected for single image operations, images are input and output from the insert point with standard screen scrolling. When selected for multi-image mode operation, tape blocks are input and output from home position and each line represents an image.

Error recovery for the TCS 610 includes detection of end of tape in which case output is held. When a data error cannot be recovered by the CCR, output is held and the message "DATA ERROR - OUTPUT HELD" is sent to the terminal operator.

8.2.2.2. Operational Considerations for the DCT 1000

The general operating procedures for the DCT 1000 are very much the same as those for the UNISCOPE 100/200 Display Terminal (see 8.2.2.1) with the following additional considerations:

1. Initialization Procedure:

   a. Set switches to the following positions:

      AUTO/MAN - AUTO
      MONITOR ON/OFF - ON
      ON LINE/OFF LINE - ON LINE
      KEYB'D/OFF - KEYB'D
      ALL OTHER DEVICE SWITCHES - OFF

   b. Establish line connection.

   c. Press clear key and set RUN/STOP switch to stop position and then run.

   d. Enter the 6-character site-id for this terminal from the keyboard.

   If the transmitted site-id is valid for this particular terminal and is received properly, the standard message is printed at the terminal. The greater than sign, (>), is returned as a solicit and indicates that the DCT 1000 is ready for input. If the message is not received properly by the central site, no message is sent. The operator should then validate the site-id entered and repeat the initialization sequence.

2. When not in queued mode (see 8.2.2.3), the operator should always wait for a solicit character > and the green keyboard ready light before striking any keyboard keys, since failure to wait could cause the terminal hardware to be unable to receive from the central site. This could lead to termination of the terminal, at the discretion of the console operator.

3. The terminal operator must inform the symbiont by a control statement, (see 8.2.2.3) of the output device to be used. Output devices need not be selected by switches since they are selected by the symbiont.

4. Termination of the DCT 1000 is identical to that of the U100/U200.

5. A feature applicable only to the DCT 1000 is point-to-point (PTP) mode (formerly known as hardware batch mode). PTP uses an alternate communications discipline to achieve higher

throughput to a specific terminal. However, it also temporarily dedicates the line to that terminal and, hence, degrades the performance of all other terminals on that line. Terminal operation is essentially the same for PTP mode as for normal mode.

6. All demand runs submitted via the DCT 1000 must have their @RUN control statements entered via keyboard. All @RUN control statements entered via cards or the paper tape reader initiate batch runs.

8.2.2.3. Operator Screen and Input/Output Control Statements

All the demand symbiont control statements (see 8.1.1.3.1) are available to the U100/U200 or DCT 1000. This subsection contains a description of the operation of these symbiont control statements.

1. The following may be requested for the UNISCOPE 100/200 Display Terminal:

   a. @@END FUL – End full screen input mode.

   b. @@FUL – Enable full screen input (queued mode). A temporary mass storage file is assigned to contain queued multiple images and the message **FULL SCREEN** is displayed. Thereafter, multiple images, up to a full screen at a time, may be transmitted. Images in each transmission are spooled to the temporary file. When @@END is transmitted, the file is closed. Subsequent READ$ requests are honored by images in the file until an end-of-file is reached. The temporary file is then freed and normal input mode is resumed.

   c. @@INS xx – Set screen insert point at xx, where xx is the number of the line on the screen. If xx is omitted, the original system setting specified at system generation is used.

   d. @@NOPR – End @@PRNT mode.

   e. @@PRNT xx – Start printer on Quiet Terminal Printer (QTP) number xx. An error message is returned if the QTP is not configured or cannot be selected. If there is more than one QTP, operands are assigned at system generation and are mandatory, as the @@PRNT is ignored without one. If there is only one QTP, the xx may be left out. Up to six QTPs can be configured for each U100/U200.

   f. @@RLD – Set screen roll direction to down. There is no response to this message.

   g. @@RLU – Set screen roll direction to up. There is no response to this message.

   h. @@TCI unit,number – The specified tape cassette unit is selected for input. Input is continuous until either the specified number of blocks have been read or a tape mark (@@TCM) is read. Input operations are single image unless requested while in full screen mode (@@FUL).

When inputting blocks in single image mode, the insert point may have to be changed (@@INS). The setting must allow enough lines to input the block plus two extra; otherwise, the cursor may wrap around to top of screen prior to the transmit function and data is lost.

When inputting data in multiple image (@@FUL) mode, each block is read in at the top of screen and each line represents an image.

NOTE:

AUTO TR switch must be set when inputting to allow data to be transmitted.

i. @@TCM – @@TCM is a file tape mark. A tape mark input from the cassette initiates termination of input operations and deselection of the unit.

If the tape cassette is output selected, an @@TCM control message initiates the writing of a tape mark and deselection of unit.

@@TCM must be written as a single image block to represent a tape mark.

j. @@TCO unit, margin – Select tape cassette unit for output. In the absence of margin, output formatting is single image mode with one image per block written from the insert point, followed by scrolling.

The margin can be set to indicate the maximum number of lines to output per block for multiple image blocks. Margin is limited to the screen size (12, 16, or 24). The screen is erased after each block is written. Tapes with multiple image blocks must be inputted in @@FUL mode.

k. @@TCT unit,track – Position specified tape cassette unit to top of track 1 or 2. Track 1 is the default value. The unit is deselected following the operation. It is advisable to rewind to top of track via @@TCT prior to other operations to ensure the unit is configured and the cassette is seated properly on the transport.

2. Requests valid for either DCT 1000 or UNISCOPE 100/200 terminals are:

a. @@END – End special mode. Return terminal from any of the special modes (escaped and queued) to normal input mode.

b. @@END ESC – End ESCAPE MODE

c. @@ESC [I] [O] – The I option enables escape mode for input. Cursor positions are not stripped from the input data while in escape mode.

For example, if a terminal operator were to @XQT the program, and then @@ESC, the program could determine the cursor positions by examining the input data returned via an ER AREAD$.

The O option enables escape mode for output. The user must then send correct images via ER APRINT$.

See UNIVAC UNISCOPE 100 Display Terminal Programmer Reference, UP-7807 (current version) for the meaning of the cursor control characters.

3. DCT 1000 Requests

a. @@END CDI – End card input mode and begin processing the mass storage buffered input.

b. @@END PTI – End paper tape input mode and begin processing the mass storage buffered input.

c. @@PRNT – Select printer for output.

d. @@PTI/@@CDI – Select queued mode and enable paper tape or card input. The DCT 1000 must be configured with a tape reader or card reader. One of two messages may be returned to the terminal indicating that either the request for queued mode was rejected or accepted.

If queued mode is accepted, the operator must:

- clear the KEYBOARD switch;
- set the paper tape reader or card reader switch; and
- throw the RUN switch to STOP, then to RUN.

The input begins. An @@END must be submitted after the input to allow the queued output to be sent.

e. @@PTO or @@CDO – Enable paper tape or card output. This message type allows for computer selection of the appropriate output device. An @@PRNT must be entered following all punching to redirect output to the printer.

The operator may override the output device selected by the demand symbiont control statement. Unless he chooses to do so, no switch settings are necessary.

f. @@PTP – Enter point-to-point mode.

4. UTS 400 Requests

a. @@END POC – Used to terminate a power-on confidence mode in which a UTS 400 master terminal may be a result of a previous @@POC command initiated from that particular UTS 400 master terminal.

b. @@ERLG – When initiated from a UTS 400 master terminal, the host processor requests the error log of the UTS 400 terminal system. The UTS 400 responds with a text message which contains the error log data. See SPERRY UNIVAC Universal Terminal System 400, Programmer Reference, UP-8359 (current version). The host processor passes this text on to the Series 1100 user program associated with the master terminal. The host processor then sends a clear error log command to the UTS 400 terminal system. The UTS 400 resets the error log to zero.

The @@ERLG command initiated from any terminal other than a UTS 400 master is illegal and the "ILLEGAL @@IMAGE FOR THIS DEVICE" error message will occur.

c. @@PMOD *x* – Used in conjunction with the @@TCO or the @@PRNT commands. See SPERRY UNIVAC Universal Terminal System 400, Programmer Reference, UP-8359 (current version) for complete definition of the Peripheral Mode operations. The @@PMOD command is terminated or returned to Print Transparent mode (normal mode) by entering @@TCM or @@NOPR.

Where *x* can be one of the following options:

V – transfer variable
A – transfer all
C – transfer changed
P – print all
F – print form
X – print transparent

d. @@POC – When initiated from a UTS 400 master terminal, places the UTS 400 master terminal in a power-on confidence mode (POC). While in POC mode, the DLE 6 status response passed by the UTS 400 at the completion of a confidence test, if strapped to do so, passes on to the Series 1100 user program active on the UTS 400 master terminal.

The @@POC command initiated from any terminal other than a UTS 400 master terminal is illegal, and the message "ILLEGAL @@IMAGE FOR THIS DEVICE" will be generated.

In POC mode, a UTS 400 master terminal is terminated with either an @@END POC, @@END, or an @@TERM command.

e. @@SCBY *stat-id* – Where *stat-id* is the terminal-id (Terminal Descriptor ID) for the Screen Bypass Terminal (SBT). Allowed only on UTS 400 master or slave terminals which are connected to the same cluster as the SBT. The terminal which initiates this command enters Screen Bypass Mode (SBM). All input after entering SBM is entered as if initiating from the SBT until SBM is terminated (see @@TSBM).

f. @@TSBM – Exits the terminal which was initiated from SBM. This allows the initiated terminal to return to normal demand mode while the SBT continues to perform the runstream originally initiated.

8.2.2.4. Operational Considerations for the UTS 400

The UTS 400 is a programmable terminal which can be downline loaded; that is, take an absolute element which has been created to execute specific functions within the UTS 400 and load the element through the communications line from the Series 1100 System to the UTS 400 storage. The UTS 400 can be used as a single station or as a cluster configuration. A cluster configuration is defined as a UTS 400 master terminal which has at least one UTS 400 slave attached to it, or a UTS 400 controller which has at least two slaves attached to it. The UTS 400 master can have up to 2 slaves attached to it. The UTS 400 controller can have up to 6 slaves attached to it, with 1 of the 6 slaves designated as the primary slave. The primary slave will be referred to in this document as the master terminal on controller type clusters. The master cluster or the controller cluster may have one (1) Screen Bypass Terminal (SBT) feature attached to it in place of one slave terminal. Only a UTS 400 master can request a downline load operation.

The UTS 400 has a total of 22 function keys, which can be meaningful to the user programs within the UTS 400, or be transmitted to a user program within the Series 1100 System.

The UTS 400 has Peripheral Sharing; that is, a cluster of UTS 400s can share the use of a COP, Terminal Printer 800, an enhanced Tape Cassette 600, and a Diskette Subsystem. However, only one terminal at a time can access a particular peripheral device while operating online to the Series 1100 System. The UTS 400 has its own auto error recovery for peripheral device errors.

Capabilities to receive or generate Field Control Characters (FCCs) can be initiated by the UTS 400. The different types of FCCs can create blinking fields, low intensity fields, normal intensity fields, tab stops, etc., on the screen of the UTS 400.

8.3. TERMINAL SECURITY SYSTEM (TSS)

8.3.1. General

The concept of "logging-on" a time-sharing system or being validated before scheduling occurs in a batch oriented system is derived from one or more motives. Primarily, it assures that every user running in demand or batch mode is a valid user of the system. It also relieves the demand user of the task of providing information to the Executive via the @RUN image.

The basis of TSS is the configuring or identification of all users. This allows the Executive to know enough about a user when the user enters the system to perform the required validation. Mechanically, this involves the user submission of USER-ID (identifying the user) and a PASSWORD (verifying the USER-ID). From this simple log-on procedure, a wide range of variations may be obtained, depending on how the user is configured and how the system is configured.

8.3.2. Demand Mode Logon Modes

The user must be configured to one of three different modes of logon: basic mode, run mode, and execution mode.

If the user is in run mode or execution mode, TSS informs the user during the logon process about the run number and when the last run occurred.

RUN NUMBER XXX is a cumulative decimal value indicating how may runs the user has executed. LAST RUN AT: *mmddyy hhmmss* indicates the last date and time the user entered the system.

8.3.2.1. Basic Mode

If configured for basic mode, the user always enters @RUN image. In basic mode, the user logon process requires entering a SITE-ID, a USER-ID, and a PASSWORD. The SITE-ID depends on the terminal which is being used. The USER-ID and PASSWORD are assigned to the user by the installation. In this example, the USER-ID is JONES and the PASSWORD is HAPPY.

When JONES logs on the system, the following occurs:

1. Jones enters SITE-ID:

   SU1801

2. The system responds with:

   ENTER USER-ID/PASSWORD

3. Jones must then enter:

   JONES/HAPPY

   followed by a TRANSMIT or carriage return.

4. The system responds with:

   DESTROY USER-ID/PASSWORD ENTRY

   UNIVAC 1100 OPERATING SYSTEM.

5. Jones is now logged on the system and may enter an @RUN image.

NOTE:

When entering an @RUN image in demand mode and the TSS file is initialized, the user-id subfield of the account field is optional. If the TSS file is not initialized, the user-id must be supplied on the @RUN statement. If the user-id field on the @RUN image is not used, the user-id supplied initiated at logon time is used. However, if user-id subfield is used on the @RUN image, it must be the same as that used at logon time or the run is rejected and the message "USER-ID DIFFERENT FROM LOG-ON" appears.

8.3.2.2. Batch Run From a Demand Terminal

The RUN,/B option specifies that this run should be scheduled as a batch run (see Table 3–2). As a result, this run is required to follow the rules of batch password validation (see 8.3.2.6). Even though the user has signed on with a valid password, the @RUN stream must also contain an @PASSWD card or the run is rejected.

8.3.2.3. Run Mode

Run mode is similar to basic mode in that the user is required to enter a USER–ID and a PASSWORD. If the user is configured for the run mode, a @RUN image is generated by TSS for the user unless the optional asterisk (*) precedes the user–id. In this case, the user is allowed to enter an @RUN image. In the following examples, the RUN–ID generated for the user on the @RUN card is the USER–ID entered at log–on time. In run mode, the user log–on process is required to enter a SITE–ID, a USER–ID, and a PASSWORD. The SITE–ID depends on the terminal being used. The USER–ID and PASSWORD are assigned to the user by the installation. In this example, the USER–ID is SMITH and the PASSWORD is REDDOG.

Example:

```
SU1801
ENTER USER-ID/PASSWORD
SMITH/REDDOG
DESTROY USER-ID/PASSWORD ENTRY
UNIVAC 1100 OPERATING SYSTEM
RUN NUMBER 5
LAST RUN AT:  121274  111515
DATE:  121374  TIME:  094632
```

NOTE:

In the above example, the asterisk was not used. If it had been used, the following would have occurred if the user was configured to enter a run image.

Example:

```
SU1801
ENTER USER-ID/PASSWORD
*SMITH/REDDOG
DESTROY USER-ID/PASSWORD ENTRY
UNIVAC 1100 OPERATING SYSTEM
DATE:  121374  TIME:  151223
```

The user would now be able to enter an @RUN image.

Other variations of the above sequence allow the user to enter the account number and/or project–id if configured by the site to do so.

Example:

```
SU1801
ENTER USER-ID/PASSWORD
EFX/SHADOW
DESTROY USER-ID/PASSWORD ENTRY
```

```
UNIVAC 1100 OPERATING SYSTEM
ENTER ACCOUNT
599210
ENTER PROJECT-ID
DSYS4
RUN-NUMBER 12
LAST RUN AT:  121074  080211
DATE:  121374  TIME:  011531
```

Another example illustrates how a user chooses an account number if configured for more than one. A user may be configured for up to and including five account numbers. The response to the "CHOOSE ACCOUNT INDEX" specifies the first, second, or third, etc., account number desired.

Example:

```
SU1801
ENTER USER-ID/PASSWORD
EEA/12A
DESTROY USER-ID/PASSWORD ENTRY
UNIVAC 1100 OPERATING SYSTEM
CHOOSE ACCOUNT INDEX
3
RUN NUMBER 10
LAST RUN AT:  120974  030506
DATE:  121374  TIME:  181654
```

8.3.2.4. Execution Mode

Execution mode is similar to run mode; however, it differs in that it places the user in an execution mode (e.g., Conversational Time Sharing – CTS) after the @RUN image is automatically built by TSS. The execution mode imposed on the user is dependent on how the user is configured for the installation. Each time an @FIN is done on a run and another run is started, the user is placed in the configured execution mode. If a user is configured for execution mode, the following example illustrates the sequence which occurs.

Example:

```
SU1801
ENTER USER-ID/PASSWORD
BCD/D12
DESTROY USER-ID/PASSWORD ENTRY
UNIVAC 1100 OPERATING SYSTEM
RUN NUMBER 4
LAST RUN AT:  120274  164212
DATE:  121374  TIME:  094234
CTS 7R1  12:45:10 (CTS processor-id line)
```

8.3.2.5. System Contingencies

TSS can be configured by the installation to respond in various forms when an illegal user-id and/or password is entered. The resultant actions are as follows:

1. The user may be allowed to run;

2. terminate the user's terminal;

3. or the user may be resolicited for the user-id/password by the message "ENTER USER-ID/PASSWORD".

In addition to the above described actions, the terminal message "ID NOT ACCEPTED" may appear if the installation is so configured.

8.3.2.6. Password Modification (@@PASSWD)

The user is provided the ability to change the password in batch and demand mode. In demand mode, there are two ways of implementing this change. When answering "ENTER USER-ID/PASSWORD" message, the user may respond with "USER-ID/PASWD/NEWPASS" where NEWPASS is the new password. TSS changes the password from PASWD to NEWPASS. In demand mode, the user may change the password while the run is active through the use of the @@PASSWD control statement.

Example:

```
@@PASSWD  PASWD/NEWPASS
```

Successful replacement of the new password causes the message "PASSWORD REPLACED" to be displayed at the terminal. If the password, supplied as the current password, is not legal, "ID NOT ACCEPTED" is displayed if the installation is so configured and one of the system contingencies occurs (see 8.3.2.5).

In batch mode, the user may change the password by using @PASSWD control statement.

Format:

```
@PASSWD  OLDPSW/NEWPSW
```

Use of the @PASSWD control statement changes the user's password from OLDPSW to NEWPSW.

If the password, supplied as the current password, is not legal, the run is removed and the message "RUN REMOVED DUE TO SECURITY ERROR" is placed in the run's print file.

8.3.2.7. Resolicitation of User-id and Password

In demand mode, each time a run fins (@FIN), an automatic resolicitation of the user-id and password occurs. When a run is @FINed and the user wishes to start another run from that terminal, an @@CONT control statement causes the "ENTER USER-ID/PASSWORD" message to appear. After an @FIN, only an @@CONT, @@TERM, or @@SEND is allowed. Any other control statement causes the "ENTER USER-ID/PASSWORD" message to appear, as does the @@CONT.

8.3.3. Use of TSS Processor

The majority of the TSS processor commands are privileged and designed for installation use. However, the commands, MAXTIME, MAXPAGE, LIST, RESET, and EXIT are unprivileged commands available for the user. MAXTIME and MAXPAGE can be used to change the system standard values of max time and max pages provided the user when automatic run card generation is configured (Run Mode and Execution Mode) for the user. EXIT causes the TSS processor to exit. RESET can be used to reset the RUN NUMBER to zero (see 8.3.2). The maximum value for RUN NUMBER is 999. LIST can be used to list the information such as project-id, account number(s), max time, max pages, last run number, console mode, message mode, etc., about the user-id.

Example:

```
MAXTIME USER-ID,   VALUE
MAXPAGE USER-ID,   VALUE
RESET USER-ID
LIST USER-ID
EXIT
```

The TSS processor is invoked by @TSS.

Example:

```
@RUN TSS,123456/USER1,TSS
@PASSWD  PASS1
@TSS
MAXTIME  USER1,60
MAXPAGE  USER1,500
RESET  USER1
LIST USER1
EXIT
•
•
•
@FIN
```

8.3.4. Error Messages

The following error messages may be returned on systems using TSS:

ID NOT ACCEPTED

A user-id or password entered by the user is illegal.

MASTER OR SUBMASTER KEY IN ERROR

The user has attempted to use a privileged instruction (presented by TSS).

MISSING USER-ID

The user has not specified a user-id on the account subfield on the @RUN card in batch mode.

@PASSWD STATEMENT IGNORED

The user has placed an @PASSWD statement in an @ADD file.

RUN REMOVED DUE TO SECURITY ERROR

The user's run has been removed because either an illegal password or no password was found in the runstream.

USER-ID DIFFERENT FROM LOG-ON

A user-id has been entered on the @RUN card which is not the same as that used at log-on time.

8.4. CONSOLE CAPABILITY

There are four levels of console capability available to a demand terminal operator. These capabilities or modes are called Basic Keyin, Limited Keyin, Full Keyin and Display, and Keyin. Each mode is described below:

- Basic Keyin

  This mode allows a user at a demand terminal to manipulate or request status on the user's run, on any run started by that user, or on any run containing a user-id which matches the user-id at the terminal.

- Limited Keyin

  In this mode, the demand terminal user has use of all the keyins specified in Basic Keyin mode plus the capability of using many of the status keyins.

- Full Keyin

  A demand operator with Full Keyin console capability has use of all the unsolicited keyins except the following:

  AC, BP, C, CK, CP, DU, ET, FB, FC, II, IT, LB, LC, RL, RP, RS, SB, TJ, TP, TR, and $! keyins. Figure 8-2 illustrates the configured keyin to console mode relationship for all keyins.

- Display and Keyin

  This mode not only gives a demand terminal Full Keyin capability, but it also allows a specified number of message groups to be displayed at that terminal. The following restriction applies to the number of message groups which may be displayed at a demand terminal. All, or from 1 to 10, message groups can be displayed at a terminal. An operator at a demand terminal which is in Display and Keyin console mode, with the following exceptions, has all the capabilities which an onsite console operator has. The limitations are as follows:

  1. The terminal cannot be a boot console.

  2. The terminal cannot be a response console for a message group nor is the operator allowed to use an AC keyin to make the terminal a response console.

3. The continuous display is not sent to any terminal in this mode.

4. No special scrolling procedures is used for messages coming to terminals in this mode.

TSS is the vehicle for specifying console capability. User-ids or user-id/site-id combinations are granted console capability when TSS file is set up or updated.

For all console modes, console capability at a terminal is requested by means of the @@CONS control statement. Two variations of this control statement may be used. When a series of unsolicited keyins are to be typed in, the following sequence must be used:

```
@@CONS
Keyin 1
Keyin 2
   •
   •
   •
Keyin n
@@END CONS
```

The @@END CONS closes out the console capability.

The second variation of the @@CONS control statement is illustrated below:

```
@@CONS Keyin
```

This format is used when only one keyin is to be typed in.

For those users who initiate Display and Keyin mode with an @@CONS statement, an option letter exists to permit them to inhibit automatic displaying of messages. The option letter is K, and the control statement with the option letter appears as follows:

```
@@CONS,K
```

The rules for handling all input and output upon entering, while in, or after leaving a console mode, are as follows:

1. All console mode input is ignored after a console mode has been terminated. This input consists of unsolicited keyins and responses to Type and Reads resulting from the processing of unsolicited keyins. Typing in an @@END CONS terminates console mode when the following sequence is used:

   ```
   @@CONS
   Keyin 1
      •
      •
      •
   Keyin n
   @@END CONS
   ```

   In the absence of an @@END CONS, the user stays in console mode until an @@TERM is typed in. When a user types in a

   ```
   @@CONS Keyin
   ```

statement, console mode is terminated immediately after the keyin has been accepted. Thus, if this statement contains any keyin which results in a Type and Read such as a DN keyin, the Type and Read is not displayed at the terminal and, thus, the user is not allowed to respond to the Type and Read.

2. Termination of Display and Keyin mode also causes the termination of all message groups going to that terminal. If the user initiates Display and Keyin mode via a

   @@CONS *Keyin*

   statement, there is no display of message groups at that terminal since, in this case, console mode is terminated before routing of the message groups is set up. Thus, the use of the K option on this statement, while not illegal, really has no meaning; since when this keyin is used, display mode is never initiated.

3. Type and Read messages and responses or lack of response to these messages are handled in the following ways:

   a. If a Type and Read message occurs after the user has terminated console mode, the message along with a header message is sent to an onsite console. The format of the header is as follows:

      RESULT OF KEYIN FROM *site-id*

   b. Once a Type and Read message has been displayed at a terminal, no additional input besides a response to that message is allowed from that terminal. When input comes in which is unacceptable as an answer, the following message is displayed at the terminal:

      ANSWER MESSAGE

      This is followed by a redisplay of the Type and Read message. The cycle of accepting input, verifying it as a response, and sending out the reject message along with the original Type and Read when the answer is unacceptable, continues until the user answers the message, the terminal times out, the user goes out of console mode, or the user types in @@TERM.

      The Type and Read message sent to a terminal always has the format:

      0 *Type-and-Read-message*

      If the response does not conform to the following two conditions:

      1. a format of "0 answer", and

      2. the number of characters in the answer is between 1 and the maximum allowed.

      it is considered unacceptable. If the input is acceptable, it is sent along to the routine which issued the Type and Read message and the console handler deletes the original Type and Read message. It is then up to the keyin routine to determine if the response is correct, and to reissue the message if the answer is not correct.

4. All console output (results from keyins, display of message groups, and Type and Read messages from the keyin elements) is susceptible to being discarded when it builds up. In all cases, a warning message is sent to the terminal informing the user of the loss of console output. When the user terminates console mode, all passing of message groups and Type and Read messages to the terminal ceases. The remaining type of output (normal output from a keyin routine) continues to be displayed until the keyin routine sends out the last line of output.

NOTE:

There is no attempt to segregate print output from console output while in console mode, or console output from print output once console mode has terminated.

Table 8-2. Keyins and Console Modes Allowed

	Keyin	Console Mode
1.	B <X>	Full Keyin
2.	CS A CS H CS ALL	Full Keyin
3.	CS AT CS AD CS HT CS HD	Full Keyin
4.	D	Basic Keyin
5.	DN IN MV RV SU UP	Full Keyin
6.	E <run-id>	Basic Keyin(5)
7.	FS (All)	Limited Keyin
8.	GO SM (All)	Full Keyin
9.	LG * <run-id> <text>	Basic Keying(5)
10.	LG <text>	Full Keyin
11.	LS (All)	Full Keyin
12.	PC (All)	Full Keyin
13.	RC <run-id>	Basic Keyin(3)
14.	RM <run-id> CS H <run-id> CS A <run-id>	Basic Keyin(5)

Table 8-2. Keyins and Console Modes Allowed (continued)

	Keyin	Console Mode
15.	SQ	Limited Keyin
16.	SQ <run-id> <file> QTO <sname2> SQ <run-id> <file> QTO <user-id>/U	Basic Keyin(8)
17.	SQ <run-id> P SQ <run-id> <file> P	Full Keyin
18.	SQ <run-id> *R	Basic Keyin(7)
19.	SQ <sname>	Basic Keyin(1)
20.	SQ <sname> *	Basic Keyin(2)
21.	SQ <sname1> QTO <sname2>	Basic Keyin(4)
22.	SQ <sname1> TO <sname2>	Basic Keyin(4)
23.	SQ <user-id> *U	Basic Keyin(6)
24.	SQ USR*ID SQ USR*ID *	Limited Keyin
25.	SR ,<unit>/<reel> A/F	Full Keyin
26.	SR ,<unit>/<reel> A/F <sname>	Full Keyin
27.	SR ,<unit>/<reel> <run-id1> ,..., <run-idn>	Full Keyin
28.	SR ,<unit>/<reel> <run-id> <file>	Full Keyin
29.	ST	Full Keyin
30.	SV ,<unit>/<reel> A/F	Full Keyin
31.	SV ,<unit>/<reel> A/F <sname>	Full Keyin
32.	SV ,<unit>/<reel> <run-id1> ,..., <run-idn>	Full Keyin
33.	SV ,<unit>/<reel> <run-id> <file>	Full Keyin
34.	SX A/F	Full Keyin
35.	SX A/F <sname>	Basic Keyin(4)
36.	SX <run-id1> ,..., <run-idn>	Basic Keyin(8)
37.	SX <run-id> <file>	Basic Keyin(8)

Table 8-2. Keyins and Console Modes Allowed (continued)

	Keyin	Console Mode
38.	T T H BL SS	Limited Keyin
39.	TB <text>	Full Keyin
40.	TM <site-id> <text>	Limited Keyin
41.	UR ; UR <sname> ; UR <site-id>	Limited Keyin
42.	UR <site-id> T	Full Keyin
43.	UR <site-id> <message>	Full Keyin
44.	X <run-id>	Full Keyin

NOTES:

1. *Basic Keyin only if <sname> matches site-id; otherwise, restricted to Limited Keyin mode.*
2. *Basic Keyin only if <sname> matches site-id and then only for output queue entries that have user-ids matching the current user-id at the terminal; otherwise, restricted to Limited Keyin.*
3. *Basic Keyin only if user-id of <run-id> matches user-id at terminals; otherwise, restricted to Limited Keyin.*
4. *Basic Keyin only if <sname1> matches site-id and then only for output queue entries that have user-ids matching the current user-id at the terminal; otherwise, restricted to Full Keyin.*
5. *Basic Keyin only if user-id of <run-id> matches user-id at terminal; otherwise, restricted to Full Keyin.*
6. *Basic only if <user-id> matches currently active user-id at terminal; otherwise, restricted to Limited Keyin.*
7. *Basic Keyin only for those output queue entries with <run-id> that also have user-ids matching the current user-id at the terminal; otherwise, restricted to Limited Keyin.*
8. *Basic Keyin only for those output queue entries with <run-id> that also have user-ids matching the current user-id at the terminal; otherwise, restricted to Full Keyin.*

8.5. TERMINAL USER TECHNIQUES

The following suggested techniques will more effectively utilize the system:

1. Do not type directly into batch-type processors such as COBOL, FORTRAN, ASM, ALG, MAP, etc. Introduce line corrections by the @ADD command, or use ED on CTS to make source corrections.
2. Minimize listing output even to the central site. The N option should be used on the @RUN control statement entered from remote terminals. This inhibits postmortem and dynamic diagnostic dumping.

3. Do not initiate long-running, compute-bound programs while in demand mode. Use the @START, @@START, or B option on the @RUN control statement (@RUN,/B). This not only improves system scheduling, but also frees the terminal for further use.

4. Make the text of messages as concise as possible.

5. Output via the @SYM statement (after the desired output is in a file) rather than SITE from CTS or ED, which would tie up the terminal.

6. Do not unnecessarily use the U option on the @SYM statement since it inhibits decataloging of the file when processing is completed (applicable only to user-defined files). This loads the system with unnecessary files. Caution should be used to avoid using the file before printing is completed.

7. Use @ADD to include canned (cataloged) runstreams for repetitive functions.

8. Use @COPY,G statement instead of @COPIN,@COPOUT operations to tape. The @COPY,G is faster.

9. Use the @CAT rather than @ASG,C when cataloging. The @CAT control statement is used to catalog files without having them assigned to the run.

10. Free tape drives as soon as possible.

11. @FREE or @DELETE cataloged files as soon as possible. This makes the assigned space available for other uses.

12. Pack files often to release space used by deleted elements.

13. Initial reserve should only be used in very specialized circumstances.

14. The @@ASG of a removable disk does not tie up a terminal until the facility becomes available, and is an easy way to have disk packs mounted.

15. Operations which alter the directory information are not to be done until the disk pack is registered.

16. Keep disk packs mounted for a minimum amount of time in order to make the disk drive available for other users.

17. Use SECURE judiciously. Do not use SECURE for scratch files.

18. Instead of using several @ASG file statements, use the @ERS, and reuse the same file space whenever possible.

19. When using a processor that requires an end sentinel, input the end sentinel as part of any @ADD stream (e.g., @ADD,E). Do not input source or changes to such a processor except via @ADD.

8.6. EXAMPLE OF A DEMAND RUN

An example follows to demonstrate the use of the system in a demand processing mode from a teletypewriter.

```
U1108A  <-------------------------------------------------- User enters site-id.

ENTER USERID/PASSWORD

JONES/REDDOG

UNIVAC 1100 OPERATING SYSTEM VERS 36.R2 (RS1)*<---------- System responds.
>@RUN,B JONES,999999,APL,10,500  <---------- User: @RUN image.
DATE:  060179   TIME:  082429  <---------- System responds.
>@@TTY L,?,C,"  <---------- User has defined line deletion, ?, and character deletion,".
>@MSG032?  <---------- User makes mistake and erases line.
>@MSG JONES, 999999, NEED TAPE 428C, NO RINT"G
                                        User gets message right on second try.
                                                  Note deletion of character.
>@CAT,P JONES1,F///600  <---------- User catalogs files, system responds with READY.
READY
>@CAT,P JONES2,F///600
READY

*TM* JONES*SMITH WISHES TO CONTACT YOU<---------- System operator and terminal
>@MSG IS SMITH DOWN THERE  <---------- user converse.

>@ASG,T    TA,T,428 <---------- User assigns tape.
WAITING ON FACILITY <---------- No tape units available.
READY  <---------- Tape unit becomes available.

@@PTI  <---------- User starts paper tape by entering @@PTI control statement.
*TAPE START*<---------- System responds with message.
@COPIN TA.,JONES2.
@FREE TA
@ELT,IDL JONES2,BATCH2
@RUN,B JONES,999999,APL,10,500
@ASG,A JONES1.
@FREE TPF$
@ASG,T TPF$,F///600
@ASM,S JONES1.INP,JONES1.REL
@HDG,P EXECUTION OF APL:  RETURN TO RA JONES
@JONES1.XQT,YZ ?  <---------- Mistake, erase this line.
@JONES?  <---------- Mistake, erase this line also.
@XQT,YZ JONES1.ABS
'STRING'
'SWILLTOON'
2 2 4 $R 'ABC'
@ADD DATAPL
@END
@@END  PTI
END OF TAPE  <---------- End paper tape input.
FURPUR 28R1T1    E35 74R1Q2 09:07:18 FURPUR processes @COPIN control statement.
  12 SYM  3 REL  5 ABS
READY  <---------- Tape TA is released.
```

```
ELT 8R1 S74Q1C 04/24/79 09:16:11
000001      000   @RUN,B JONES,999999, APL,10,500
000002      000   @ASG,A JONES1.
000003      000   @FREE TPF$
000004      000   @ASG,T TPF$,F/300/TRK/600
000005      000   @ASM,S JONES1.INP,JONES1,REL
000006      000   @HDG,P EXECUTION OF APL: RETURN TO RA JONES
000007      000
000008      000
000009      000   @XQT,YZ JONES1,ABS
000010      000   'STRING'
000011      000   'SWILLTOON'
000012      000   2  2  4  $R  'ABC'
000013      000   @ADD DATAPL <---------- Paper tape input is completely processed.
>@PRT,T JONES2. <------------------------- User lists contents of a file element.
FURPUR 28R1T1   E35 74R1Q2 09:07:18

APL*JONES2
REL  CRELAD$
REL  SNOOPY
ABS  TDP
ABS  PCT
DOC  EDITDOC(1)
DOC  TDPDOC(1)
DOC  PCTDOC(1)
ABS  EDIT
ABS  EDREP$
ELT  BATCH1(1)
ELT  FIX(1)
ELT  ASMXQT(3)
ELT  ASMTTY(2)
ASM  ASMRUN(5)
ELT  EXECUTE(3)
ASM  ASMTB(5)
ASM  INP(2)
REL  REL
MAP  MAP(1)
ABS  ABS
ELT  BATCH2(1)

>@COPY JONES2.,JONES1. <------------------ User copies files and starts a batch run.
    89 BLOCKS COPIES
>@START JONES1.BATCH2
>@MSG JONES STARTING A BATCH RUN
>@FIN <------------------------------------------------------ User terminates.
RUNID:  JONES ACCOUNT:  999999  PROJECT:  APL <---------- System prints this
JONES*MSG:  JONES, 999999, NEED TAPE 428C, NO RING      Termination message.
JONES*MSG:  IS SMITH DOWN THERE
LOAD 428  10/6 TA  -1 JONES
JONES*MSG:  JONES STARTING A BATCH RUN
JONES FIN
TIME:    TOTAL:  00:00:01.650      CBSUPS:  001259232
           CPU:  00:00:00.600   I/O:  00:00:00.050
         CC/ER:  00:00:01.000  WAIT:  00:00:00.000
SUAS USED:  $ 3.95      SUAS REMAINING:  $9996.05
```

START: 08:24:29 JUN 1, 1979 FIN: 08:45:36 JUN 1, 1979

TERMINAL INACTIVE

Terminal ready for submission of another logon sequence or @@TERM control statement.

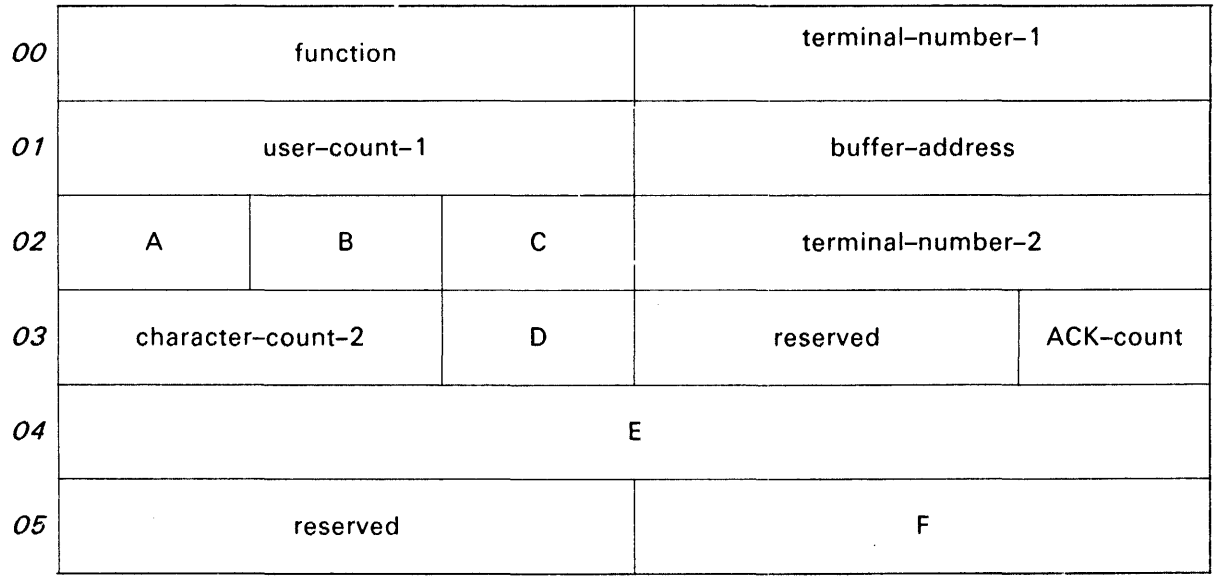
8.7. USER-SUPPLIED DEMAND SYMBIONTS (RSI$)

The general CCR (Communications Control Routines) allows a program to pass data to the Executive and receive data from the Executive in such a manner that the program is considered to be a communications control routine; i.e., a terminal handler.

The format of the call to ER RSI$ is:

```
L,U  A0,PKT-ADDR
ER   RSI$
```

where the packet has the following format:



Word 0

function – The function to be performed. These codes and their meanings are discussed in 8.7.1.

terminal-number-1 – The terminal number to which the packet applies. (See 8.7.1.1.)

Word 1

user-count-1 – This count is relative to the RSI$ function.

RSPLST$	– Maximum number of RSI$ packets in the chain.
RSCNTL$	– Number of entries in the CCR @@list.
INPUT/OUTPUT functions	– The number of ASCII quarter-word characters being passed to RSI or the number of characters which the program accepts.

buffer-address – The buffer containing (or to contain) the data being passed. This buffer must be within the storage limits of the activity performing the ER.

Word 2

A – Condition status (Octal):

040 – The ER RSI$ is in error. Check additional status information in word 4, H1 of packet for specific error.

020 – The program should request output for the terminal. This usually means that output is available; however, the program must be prepared to receive a "no-image available" status (A with 004). For RSNOT$ requests, this status applies only to print output.

010 – The system accepts input. For a demand terminal, the program should solicit input.

004 – Output was requested but none was available. For RSNOT$ requests, this status applies only to print output.

002 – The program must respond to the @@ command designated. See D, Word 3.

001 – A page eject is required for output.

B – Condition bits, for output images (Octal):

040 – The output image is the last in a series. A terminal should be prepared for user input.

020 – The output passed to the program was not complete; i.e., more output for the same image must be obtained.

010 – Special forms

004 – The output passed begins a new output image.

002 – Unused

001 – The image is the output portion of a TREAD$.

C – Condition bits (Octal):

040 – Cleared by RSI$ after the packet has been processed. See discussion of RSPLST$ function for application.

020 – ER RSI$ request cannot be satisfied due to storage limitations (EXPOOL).

010 – Punch output was requested but the "terminal" was not initialized as having a punch.

004 – Output is for the punch. For RSNOT$ requests, this bit is independent of 004 and 020 in the A-field to allow the user to determine whether print, punch, or both types of output exist.

002 – Program has attempted to pass an input record to the system for a particular terminal before a previous input record has been processed (an ER RSI$ is outstanding).

001 – Input was passed to the system via ER RSI$ before it was solicited. This is not an error but is included for program debugging purposes.

terminal-number-2 – The terminal number for which this status applies. Certain functions may return a status for any one of several terminal numbers.

Word 3

character-count-2 – On requests for output, this is the number of ASCII quarter-word characters actually passed to the program.

D – Normally, the line spacing to perform on output. However, if bit 02 is set in packet field A, this signifies which CCR-related @@ command was entered.

Possible values are:

0 – @@END

1-076 – Entry position in dynamic CCR list generated by the program via RSCNTL$.

ACK-count – In an initialization packet, the number of output images for the terminal to be sent before an acknowledge is required. If zero or 077, an acknowledge is never required. (See 8.7.1.1.)

Word 4

E – In RSBAT$, RSPUN$ and RSDEM$ initialization packets, this is a 6-character Fieldata site-id.

Additional status information for the terminal is returned in H1 of the E field on all requests. The bit(s) in this field indicate the following:

01 – The terminal has terminated. After the termination of a terminal has been initiated (@@TERM, RSTRM$/RSTRA$ Functions, "SM Site-id T", etc.), RSI$ keeps the terminal number intact until the user CCR has been notified via this status on any input/output request for the terminal, or until the run fins (@FIN).

02 – No terminals are active for this CCR.

04 – Termination is in progress for this terminal.

010 – The output passed on this request is a system "canned" message (such as *WAIT-LAST INPUT IGNORED*).

020 – The output passed on this request is from a TM or TB message.

040 – This input request was rejected by the system.

0100 – This is the last of acknowledged output for this terminal. Acknowledge is required before additional output is passed.

0200 – Lack of activity timeout warning.

0400 – Lack of activity final timeout. This timeout is final only in that the CCR continues to receive this status every timeout period until input/output is received.

01000 – The buffer specified in the RSI$ packet is outside of program limits (or RSGET$ on write protected area).

02000 – RSPLST$ error – chaining error or nested RSPLST$ request in chain.

04000 – Terminal number specified in RSI$ packet is not active.

010000 – EOF has been received for file. Output passed, if any, is part of the next file.

020000 – Illegal function code or system not configured to process this RSI$ function.

040000 – Initialization error – site-id already active, not in SMLIST (batch only), or RSI$ not allowed for the user's account/user-id quota set.

Word 5

F – Address of next RSI$ packet if this packet is in a RSPLST$ chain. Zero indicates the end of the chain.

8.7.1. ER RSI$ Functions

Functions which may be placed in the ER RSI$ packet are divided into the following groups:

1. Initialization Functions
2. Input Function
3. Output Functions
4. Termination Functions
5. Debugging Functions
6. General Functions

8.7.1.1. Initialization

A terminal may initialize as a demand terminal, a remote batch terminal without punch, or a remote batch terminal with punch. It should be noted that the demand terminal has batch capabilities. The initialization values are:

RSBAT$ (040) Initialize as a batch terminal without punch capabilities.

RSPUN$ (041) Initialize as a batch terminal with punch capabilities.

RSGDEM$ (044) Same as RSDEM$ but use a generic site-id.

RSDEM$ (050) Initialize as a demand terminal.

On return from initialization, terminal-number-1 has the terminal code number which is to be used by the CCR on all subsequent requests for the terminal.

For initialization via RSBAT$, RSPUN$, or RSDEM$, packet word 4 (E) must have a 6-character Fieldata site-id.

For initialization via RSGDEM$, the "classic site-id" is generated by RSI and is never known by the CCR. All files queued (PRINT$/PUNCH$/SYM) by a generic site-id are queued to the user-id entered at sign-on time unless the individual SYM specifies differently.

ACK-count may be set in the initialization packet to indicate the number of output requests to be satisfied before an acknowledge is required. If the cell is zero or 077, an acknowledge is never required. If ACK-count is set, the CCR is notified of an "ACK needed" status on the last of declared output by a status bit (see description of word 4 of the RSI$ packet). An acknowledge of output received is then required to receive additional output. There are two methods to acknowledge output:

- Any of the output functions for a specific terminal (see table 8-3). RSNOT$ is ideal for acknowledging a specific terminal when actual output is not desired on the request.

- RSAGAW$ to acknowledge all terminals.

8.7.1.2. Input Function

The input function is RSPUT$ (010). This function causes the number of characters specified to be transferred to the Executive as terminal input. If two RSPUT$ functions are outstanding for the same terminal at the same time, an error status is returned (C = 002).

8.7.1.3. Output Functions

The output functions may be classified in several ways. First, some functions always cause the calling activity to be reactivated whether or not there is something available. Other functions do not allow the activity to be reactivated until there is something to do. Functions may also be classified by whether or not they apply to a particular terminal or any terminal.

Table 8-3 shows the various capabilities of the different functions.

Table 8-3. RSI$ Output Functions

Function	Activity Regains Control Immediately	Print or Punch Output	Print Output Only	Punch Output Only	Input Solicit Causes Return (No Output)	Applicable for all Terminals	Output Acknowledge Reset
RSGET$ (020)	x	x					x
RSGPU$ (021)	x			x			x
RSGPR$ (022)	x		x				x
RSGTI$ (063)		x					x
RSGTW$ (064)		x			x		x
RSGTA$ (070)	x	x				x	
RSGAW$ (074)		x				x	
RSGIO$ (075)		x			x	x	
RSNOT$ (065)	x	x					x
RSAGAW$ (046)	x	x				x	x

Function	Description
RSGET$	Causes immediate transfer of print or punch data for a particular terminal or a "no-image" status.
RSGPU$	Same as RSGET$ but for punch output only.
RSGPR$	Same as RSGET$ but for print output only.
RSGTI$	Same as RSGET$ except that if no output is available, the request is held until output does become available.
RSGTW$	Same as RSGTI$ except that an input request also causes a return to the user.
RSGTA$	Same as RSGET$ except that the data transferred may be from any terminal associated with the run.
RSGAW$	Same as RSGTA$ except that if no output is available, the request is held until output is available.
RSGIO$	Same as RSGAW$ except that an input request also causes a return to the user.
RSNOT$	Same as RSGET$ except that no data transfer occurs.
RSAGAW$	Acknowledge all terminals for this CCR (reset all acknowledge counts).

8.7.1.4. Termination Function

The termination function is RSTRM$ (060). An "emergency shutdown" termination (RSTRA$-061) is also available. RSTRA$ causes termination of all terminals associated with the run. Termination also occurs if:

1. The terminal user enters an @@TERM image.

2. The onsite operator requests termination via a console keyin.

After the termination is complete and the user CCR has been notified of that fact (see Word 4 description of the RSI$ packet), all association between the terminal and the run ceases.

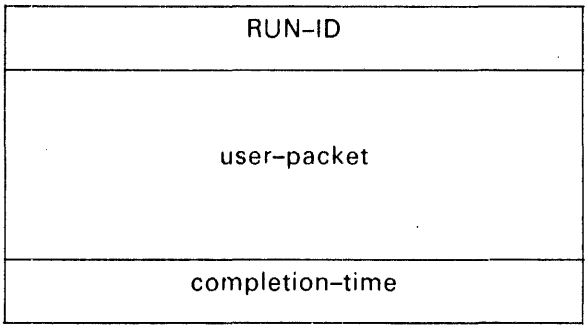
8.7.1.5. Debugging Functions

Installations have the option of generating ER RSI$ debugging functions at system generation. There are three debugging functions:

RSTRY$	(01)	Turns on internal tracing
RSTRO$	(02)	Gives trace data to program
RSTRD$	(03)	Turns off internal tracing

Several programs may use the trace data concurrently. It is important to note that a program can obtain trace data for all programs utilizing ER RSI$.

When a program executes an ER RSI$ with RSTRO$ as the function in the packet, the program must have a 60-word buffer specified. The Executive places 10 trace packets into this buffer, the first packet being the oldest. Each trace packet has the following format:



where user-packet is the first four words of the ER RSI$ packet causing the entry. The completion time is recorded in TDATE$ format.

Only 63 programs may use the trace capability simultaneously.

8.7.1.6. General Functions

RSPLST$ (045) – process the chain of RSI$ packets starting at the address specified in word 1,,H2 of the packet.

Word 1,,H1 of the packet may specify the maximum number of RSI$ packets in the chain to process. If 0, 076 is assumed. The chain link between packets is specified in word 5,,H2 with zero always taken as end of chain, regardless of number specified in RSPLST$ packet.

The user CCR may prepare a chain of intermixed RSI$ requests for any combination of terminals that it supports and present it to RSI$ via one ER. The packets are processed in a serial fashion and the user CCR does not receive control again until the entire chain is processed. The CCR may determine if an individual packet in the chain has been processed by RSI$ by setting a flag (040) into the "C" cell (2,,S3) of each RSI$ request packet. RSI$ clears the flag in each packet before going on to the next packet in the chain. This allows the CCR to react to processed packets prior to return from the ER, and to chain new packets to a chain while it is still being processed by RSI$. It also indicates whether an individual packet was processed on return from the RSI (RSPLST$) request.

The following error conditions cause RSI$ to cease processing of a chained request:

1. Nesting of a RSPLST$ request within a chain
2. Access word failure for next packet in chain
3. More than 076 in chain (to prevent looping)

RSCNTL$ (011) – the list indicated by the cell 1,,H2 of the packet is to be used as the CCR control statement list. The cell, 1,,H1 contains the number of entries in the list (62 word maximum).

Each entry is one word ASCII (alphabetic only), left-justified, space-filled. Any new CCR control statement list presented via the RSCNTL$ function when one already exists, replaces the previous one.

When processing an @@image, the RSI/CCR list is searched first. For RSI$ CCRs this would include the following:

@@X	@@SKIP
@@TERM	@@CQUE
@@CONT	@@CM
@@PASS	@@HOLD
@@INQ	@@TM
@@SEND	@@END
@@RQUE	@@CONS

If a matching entry is not found in the RSI/CCR list and a dynamic CCR list exists, it is searched next. A no find returns to normal processing – may be @@ASG, @@LOG, etc. or error condition. If a find is made, the entry number (position in the list; i.e., first entry is one) of the find is returned in cell "D" of the RSI$ packet and bit 02 is set in cell "A". "D" is zero for any @@END encountered and it is not meaningful to include "END" in any dynamic CCR list.

9. Communications Handler

9.1. INTRODUCTION

The communications handler provides the interface between the multitude of available remote terminal devices and the user programs. The diversity of hardware dictates a general routine upon which the variances of each application can be built.

Worker programs to which communications devices are assigned must be operated as real-time programs, because of the high priority which must be given to communications interrupt processing.

Each worker program to which communications devices are assigned should register an error routine with the Executive so that the worker program may be properly notified concerning operating contingencies. The error routine is registered by means of an IALL$ request (see 4.9.3) for the error mode entry. If the error routine is omitted, a contingency causes program termination.

9.1.1. Equipment

Communications devices may be connected to Series 1100 channels through four types of subsystems:

- Communications Terminal Module Controller (CTMC)
- Communications/Symbiont Processor (C/SP)
- General Communications Subsystem (GCS)
- Distributed Communications Processor (DCP)

9.1.1.1. Communications Terminal Module Controller (CTMC)

The CTMC line terminals operate in the externally specified index (ESI) mode. Each character transfer is accompanied by an address which identifies to the CPU the external line to or from which the transfer is directed, and each address has a distinct I/O access control word association. One CTMC/GCS is capable of multiplexing 64 line terminals: 32 for input and 32 for output.

The I/O operation of the 64 (0100) CTMC line terminals is controlled by ESI Access Control Words (ACWs) on the 1106, 1108, 1100/10/20 System; by the ESI ACW and Chain Pointer Word (CPW) on the 1110, 1100/40 Systems; and by the Channel Command Words (CCWs) on the 1100/80

System. For each CTMC, 0100 words (for 1110 and 1100/40 Systems, 0200 words) are reserved in low-order main storage for the ESI ACW (and CPW). On the 1100/80 System, the CCWs can be anywhere in main storage because low order main storage placement is not required due to the I/O structure.

On each I/O request from the CTMC to the processor, a unique identifier composed of the CTMC base value (bits 6 to n) and the associated CTM-id (lower 6 bits) is passed. This identifier will be used to reference the appropriate ACW or ACW/CPW pair in low order main memory for non-1100/80 Systems. For 1106, 1108, 1100/10/20 Systems, this identifier is the MSR relative address of the ACW. Due to the ACW/CPW combination on 1110, 1100/40 Systems, the hardware left shifts the identifier 1 bit. Thus, it doubles the identifier from the CTMC to reference the 2-word MSR relative ACW/CPW pair.

Any ESI channel may be configured to operate in either quarter-word or half-word mode.

The communications handler assumes that the user program is written to interface with a particular type of hardware and that the buffers are organized accordingly. The amount of time available to process characters dictates that Executive action be kept to a minimum and thus disallows character manipulation. Other than for buffer format and time considerations, the user need not be aware of the hardware arrangement being employed.

9.1.1.2. C/SP

The C/SP operates in the internally specified index (ISI) mode on any Series 1100 I/O channel. Additional information on user program interface with the C/SP is described in UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

9.1.1.3. General Communications Subsystem (GCS)

The GCS line terminals operate in ESI mode. The operation of the GCS is similar to the CTMC (see 9.1.1.1) and operates with software written for the CTMC. The GCS consists of three principal elements:

1. Communications Terminal Controller (CTC) is the multiplexing device.

2. Communications Terminal (CT) performs the communications functions: serializing, character recognition, synchronization, character parity check and generation, and dialing. The CTC can accommodate from 1 to 32 CTs.

3. Communications Interface (CI) makes the necessary conversion between the electrical operating levels of the CTs and those of the communications lines.

A GCS may be connected to any ESI word I/O channel for multiplexing up to 32 CT/CI pairs to that channel.

9.1.2. Modes of Operation

The real-time user, while interfacing with the communications handler, may operate in the following modes:

- Single Mode:

  This mode is used when one message, with a known maximum length, is to be received or sent.

- Pool Mode:

  This mode is used primarily when multiple messages are to be received or sent, or when a message of variable and unknown maximum length is to be received. There are three types of pool mode operation:

  1. Open Pool — A multiple number of buffers chained so that the chain can be exhausted. The last buffer in the chain indicates the end of the chain by a zero value in H2 of the link word.

  2. Closed Pool — For input operations only. Contains a multiple number of nonshared (1110, 1100/40, 1100/80) buffers chained in a continuous manner, where the last buffer is chained to the first; that is, the pool is never exhausted. The last buffer in the chain points back to the first by means of the link word. Use of closed pool mode requires extreme user care.

  3. Dual Pool — For input operations only. Contains an initial buffer pool for status checking, and an additional buffer pool (open or closed) for the receipt of data (not supported under C/SP operation).

The user must weigh many factors before choosing a mode of operation. Such factors are devices, carrier speed, type of acknowledgment required or not required in operating the device, and the manner in which the data is to be processed. The mode of operation need not be the same for both input and output.

9.1.3. Distributed Communications Processor (DCP Series)

The SPERRY UNIVAC TELCON System is an intelligent communications system offered as the initial implementation of Sperry Univac's Distributed Communications Architecture (DCA). Included in the TELCON System are the following:

1. SPERRY UNIVAC Distributed Communications Processors (DCP Series).

2. Network software residing in DCP Series Communications Processors.

3. DCP Series interface software residing in SPERRY UNIVAC Series 1100 Systems and Series 90 Systems.

4. Support software in SPERRY UNIVAC Series 1100 Systems and Series 90 Systems.

9.2. ASSIGNING LINE TERMINAL (LT) DEVICES

At system generation, each channel must be completely defined. For communications devices, this includes specifying the control unit type and characteristics of the Line Terminal to which the remote terminal is connected (bits per character, speed, fixed or common carrier line, unit type). At that time, various devices connected to a single line and programmed as a unit (i.e., one input and one output and/or one dialing unit) are given a line terminal group (LTG) identity. This identity can be used in assigning communications devices. Arbitrary device assignments can also be used, as specified in the @ASG control statement (see 3.7.1). For assignment of devices through a C/SP, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

9.3. THE LINE TERMINAL TABLE

The user program controls each LTG identified by means of the line terminal table (LTT) constructed in the user-program D-bank. All user operations on an LTG must reference that group through a single LTT (in contrast to other I/O devices which may have any number of active packets). The LTT enables the user program to control each LTG from the execution of the initialization request to the execution of the termination request (see 9.4.1).

The format of the LTT is divided into four parts: internal filename (words 0 and 1), output area (words 2 through 5), input area (words 6 through 9), and dial area (words 10 and 11). The user can omit any part simply by not coding it; however, omission of a part must be accounted for by a zero filled area.

The format of the LTT is:

0	internal filename				
1					
2	output-status	quarter-word-indicator	output usage	output-completion-activity-addr	output area
3	output-character-count (0 for pool mode)			output-buffer-or-pool-start-addr	
4	end-of-output-backup-queue-addr			start-of-output-backup-queue-addr	
5	partial-buffer-character-count			buffer-transfer-time	
6	input-status	end-of-input-action	input usage	input-completion-activity-addr	input area
7	input-character-count (0 for pool mode)			input buffer (pool-id for pool mode)	
8	end-of-input-backup-queue-addr			user controlled	
9	partial-buffer-character-count-or-dual-pool-addr			buffer-transfer-time	
10	dial status	dialer-selection-status	dial usage	dial-completion-activity-addr	dial area
11	dial-access-control-word				

For C/SP table variations, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

Words 0 and 1

internal filename — Contains the identity used to reference the LTG. See 2.6.2 for definition of internal filename.

Word 2

output-status — An octal code denoting the completion status of the last buffer transferred to the remote terminal. For pool mode, this code is stored in S3 of word 0 of each buffer. Values for this field are given in Table 9-1 (for C/SP output status, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

Table 9-1. LTT Output-Status Codes

Octal Code	Description
0	The number of characters specified in H1 of word 3 has been transferred to the remote terminal. For pool mode, T1 of the first buffer word has the character count.
1	The Auto Poll/Recycle Auto Poll has been terminated or a frame has been aborted by the Abort request.
2	The line has been declared down by the operator. H1 of word 5 contains the number of characters transmitted. No further action is taken on queued buffers.
4	A hardware parity error occurred during the last output. or output was terminated due to I/O path switching. If an ESI buffer timer is used, this status code is not returned for I/O path switching.
5	Output was terminated before the specified number of characters was transferred. Termination is detected by the timeout of the communications handler's timer. The number of characters transferred is placed in H1 of word 5 and the output has been turned off. This mode of operation can be used to avoid output monitor interrupts on the CTMC/GCS by specifying a full buffer, setting the EOT bit before the end of the buffer, and allowing the buffer to timeout.
10	An ESI activity associated with this line terminal is in a contingency state and, as a result, the line terminal is terminated.
20	Same as 0_8 except that the end of the buffer queue is reached and output is turned off. If a buffer is added to the queue after the communications handler makes the check, the worker program must restart the output.
24	Same as 20_8 except that output was terminated due to a hardware parity error or I/O path switching. If an ESI buffer timer is used, this status code is not returned for I/O path switching.
25	Same as 20_8 except that the output buffer has timed out.

quarter-word-indicator

For ESI activity processing, either quarter- or third-word execution is permitted. If quarter-word mode is desired, a one is placed in this field. A zero (or if system generation parameter states that quarter-word execution is not possible) indicates third-word execution. In either case, the mode is applicable to all ESI activities created with the initialization request. Once the mode is established by the real-time program, the only method to alter the mode is to terminate the line (by means of a CMT$ request - see 9.4.1.10) and reinitialize (by means of a CMS$ request - see 9.4.1.1) with the new mode. For each subsequent initialization request, the mode must be reestablished. This is required because the communications handler uses the field as a link to the program control table (PCT) item associated with the LTT after the initialization request.

output-usage

Denotes the condition required to start the output completion activity (lower two bits) and denotes the special output operation for the GCS (upper four bits). Values for the lower two bits of this field are:

0_8 – No activity is to be initiated.

1_8 – Give control to completion activity upon completion of each output buffer only if the activity is not executing. The communications handler detects exiting from this activity after the check, and restarts it.

2_8 – Give control to the output completion activity only if the output backup queue is exhausted or a nonzero status is returned for a buffer.

3_8 – This code must be used at CMS$ time to initialize the line for interrupt tabling. The interrupt tabling area ID must also be placed in the output-completion-activity-address cell. After the line is initialized for interrupt tabling, a different output usage code can be specified at CMO$ time. No completion activity activation is done if interrupt tabling is specified.

The upper four bits must be zero at CMS$ time. Values for the upper four bits at CMO$ time are:

0_8 – Normal output

1_8 – Send Local Test EF to the CT which causes the CT to switch into back-to-back (BTB) mode.

2_8 – Send Auto Poll EF to the CT which causes the CT to initiate automatic polling mode.

3_8 – Send Send Trans EF to the CT which causes the CT to send transparent text.

4_8 – Initiate Recycle Auto Poll.

5_8 – Terminate Auto Poll and Recycle Auto Poll operations.

6_8 – Abort the frame in active transmission.

(See SPERRY UNIVAC General Communications Subsystem (GCS), Programmer Reference, UP-8157 (current version) for descriptions of the GCS hardware capabilities mentioned above.)

output-completion-activity-address

For non-ESI Interrupt Tabling Operations, this field contains the starting address of the output completion activity, which is a routine to be given control upon completion of an output buffer transfer and within the conditions specified by the output usage field (S3). This activity is given control, with register A0 containing the address of the LTT, and register A1 containing the buffer address. For pool mode operations, register A1 contains the address of the first buffer transferred since more than one buffer may have been transferred prior to the activation of the completion activity. The activity specified must be within the bounds of the user program.

For ESI Interrupt Tabling operation, this field contains the interrupt tabling area ID. This ID is the value returned from a CPOOL$ request where the CPOOL$ packet specified a tabling area establishment (word 0, bit 35 = 1). All output interrupts are tabled into the tabling area during line activation. The tabling ID used in this cell must be the same as that used in the input completion activity address if input operation is to be done also.

Word 3

This word defines the single output buffer or the output buffer pool. For the CTMC/GCS (half-word operation only), output characters are transmitted in ascending order within a word starting at the lowest portion of the word. (See 9.4.2 for CTMC/GCS quarter-word transfers.) Both single mode and pool mode individual buffer sizes are limited to 4095 (2047 for 524K, 1106, 1100/10/20 systems) characters.

output-character-count — Contains the number of characters for single mode; contains zero for pool mode.

output-buffer-or-pool-start-addr — Contains the single buffer address, or the address of the buffer pool control word which is the first word of the buffer control packet. All buffers must be in the program's D-bank.

Word 4

For pool mode, this word contains the starting and ending address of the queue of buffers already filled for output to the terminal device. H1 contains the ending address, and H2 contains the starting address.

H1 and H2 must specify a buffer currently removed from the output pool via ER CGET$ and not currently in another output queue. The communications handler updates only H2. The user program can detect the end of the output queue by finding a start field with a value of zero in the LTT.

Word 5

partial-buffer-character-count — Contains the number of characters transferred as output when the output transfer of a buffer is completed before the specified count is transferred.

buffer-transfer-time — Contains the number of basic time intervals to be used as the maximum time between buffers. If a buffer does not transfer in this time interval, a fault is suspected unless the previous character denoted end of output. If a time value of zero is specified, no timing check is performed by the communications handler. It is not intended that the communications handler perform extensive buffer timing checks, but merely that it provide a means of detecting a stalled or inactive condition. Any extensive timing checks would be excessive overhead which reduces system throughput, and would be of no appreciable value to the communications handler users.

The basic time interval used by the communications handler is 600 milliseconds.

Word 6

input-status

Contains a value which denotes the completion status of the last buffer transferred from the remote terminal. For pool mode, this value is stored in S3 of word 0 of each buffer. Values for this field are given in Table 9-2. For C/SP input status, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

end-of-input-action

For pool mode, denotes action to be taken by the communications handler when an end-of-message character or external interrupt is received. Values for this field are:

0_8 - Turn input off.

1_8 - Reinitiate input using next buffer from pool.

Table 9-2. LTT Input-Status Codes

Octal Code	Description
0	Normal acknowledgment was returned. For single mode, the number of characters transferred is in H1 of word 7; for pool mode, T1 of the first buffer word contains the character count.
1	Input was terminated by an external interrupt. The number of characters transferred is in H1 of word 9 or, for pool mode, in T1 of word 0 of the buffer. *CT-HIGH LEVEL only:* I-format frame end.
2	The operator has declared the line to be down by using a DN keyin. H1 of word 9 or T1 of word 0 of the buffer contains the number of characters transmitted, if any. No further input is accepted.
3	Input was terminated before the specified number of characters were transferred, due to an input request (ER CMI$) before the previous request was completed by the handler. This applies only to pool mode; T1 of word 0 contains the partial character count, if any. Further action is specified in the end-of-input-action field which provides the user with the capability of terminating pool mode input.
4	A hardware parity error occurred during the last input (1110, 1100/40/80 only) or input was terminated due to I/O path switching. If an ESI buffer timer is used, this status code is not returned for I/O path switching.
5	A timeout was detected by the communications handler buffer timer.
6	A ring indicator external interrupt was received.
7	An external interrupt was received with a status word indicating either character or block parity error, or late input acknowledge. *CT-HIGH LEVEL only:* Frame end with FCS error.

Table 9-2. LTT Input-Status Codes (continued)

Octal Code	Description
10	An ESI activity associated with this line terminal is in a contingency state and as a result the line terminal is terminated.
11	A carrier off external interrupt was received.
12	A timeout external interrupt was received. *CT-HIGH LEVEL only:* Idle sequence detected.
13	A space-to-mark transition external interrupt was received.
16***	No idle SYNC in normal mode was detected.
17***	No idle SYNC in transparent mode was detected.
20	Same as for 0_8 except that the end of the input buffer pool has been reached; input is terminated.
21	Same as for 1_8 except that the end of the input buffer pool has been reached.
23	Same as for 3_8 except that the end of the input buffer pool has been reached.
24	Same as for 4_8 except that the end of the input buffer pool has been reached.
25	Same as for 5_8 except that the end of the input buffer pool has been reached.
27	Same as for 7_8 except that the end of the input buffer pool has been reached.
30	Same as for 10_8 except that the end of the input buffer pool has been reached.
31	Same as for 11_8 except that the end of the input buffer pool has been reached.
32	Same as for 12_8 except that the end of the input buffer pool has been reached.
33	Same as for 13_8 except that the end of the input buffer pool has been reached.
36***	Same as for 16_8 except that the end of the input buffer pool has been reached.
37***	Same as for 17_8 except that the end of the input buffer pool has been reached.
40*	ACK character was detected. *CT-HIGH LEVEL only:* S-format frame end.
41*	NAK character was detected. *CT-HIGH LEVEL only:* UN-format frame end.
42*	ENQ character was detected.

Table 9-2. LTT Input-Status Codes (continued)

Octal Code	Description
43*	EOT character was detected.
44*	DLEX character was detected.
45*	DLE0 character was detected.
46*	DLE1 character was detected.
47**	DLE EOT character was detected.
50**	EOM character was detected without being preceded by required SOM character.
51*	No response was detected from a polling operation.
	CT-HIGH LEVEL only: No response timer or auto poll stopped.
52***	ENQ character was detected in input normal text or heading block.
	CT-HIGH LEVEL only: Aborted frame.
53***	DLE ENQ character was detected in input transparent text or heading block.
54***	The normal input message had not been completed in specified time.
55***	The transparent input message had not been completed in specified time.
56***	Transparent text parity error or Late Acknowledge was detected.
57***	Transparent text no error was detected.
	CT-HIGH LEVEL only: I-format frame end with partial byte.
60*	Same as for 40_8 except that the end of the input buffer pool has been reached.
61*	Same as for 41_8 except that the end of the input buffer pool has been reached.
62*	Same as for 42_8 except that the end of the input buffer pool has been reached.
63*	Same as for 43_8 except that the end of the input buffer pool has been reached.
64*	Same as for 44_8 except that the end of the input buffer pool has been reached.
65*	Same as for 45_8 except that the end of the input buffer pool has been reached.
66*	Same as for 46_8 except that the end of the input buffer pool has been reached.
67**	Same as for 47_8 except that the end of the input buffer pool has been reached.
70**	Same as for 50_8 except that the end of the input buffer pool has been reached.

Table 9-2. LTT Input-Status Codes (continued)

Octal Code	Description
71*	Same as for 51_8 except that the end of the input buffer pool has been reached.
72***	Same as for 52_8 except that the end of the input buffer pool has been reached.
73***	Same as for 53_8 except that the end of the input buffer pool has been reached.
74***	Same as for 54_8 except that the end of the input buffer pool has been reached.
75***	Same as for 55_8 except that the end of the input buffer pool has been reached.
77***	Same as for 57_8 except that the end of the input buffer pool has been reached.

* May occur on the CTA-ISO, CTS-ISO or CTS-TRANS
** May occur on the CTA-ISO, or CTS-ISO
*** May occur on the CTS-TRANS

NOTE:

Ring Indicator and Carrier Off interrupts may occur whether or not input is active. If either of these interrupts occurs and input is active, normal processing of the interrupt is performed with the buffer or data address corresponding to the interrupt passed in register A1 to the ESI activity along with the appropriate status code. If either of the above interrupts occurs and input is not active, register A1 is set to zero to indicate the abnormal condition and is passed to the ESI input activity. The appropriate status code is provided in the input status field of the corresponding LTT for the inactive input condition.

input-usage

Denotes the conditions required to start the input completion activity on noninterrupt tabling CMI$ requests or to initialize the line for interrupt tabling on a CMS$ request. Values for this field are:

- 0_8 – No activity is to be initiated.
- 1_8 – Give control to the completion activity only if the activity is not executing. (The communications handler detects exiting from this activity after the check and restarts it.)
- 3_8 – This code must be used at CMS$ time to initialize the line for interrupt tabling. The interrupt tabling area ID must also be placed in the input-completion-activity-address cell. The tabling area ID must be the same as that placed in the output completion activity address if both input and output sections of the line are to be initialized.

input-completion-activity-address

For non-ESI Interrupt Tabling operation, this field contains the address of a routine to be given control upon completion of an input buffer transfer as indicated by the input usage field. Control is given to this routine with register A0 containing the address of the LTT, and register A1 containing either the address of the first buffer transferred (pool mode) or the address of the user's data area (single mode). For pool mode, more than one buffer may have been transferred prior to giving control to the completion activity (see word 8).

For ESI Interrupt Tabling operation, this field contains the interrupt tabling area ID. This ID is the value returned from a CPOOL$ request where the CPOOL$ packet specified a tabling area establishment (word 0, bit 35 = 1). All input interrupts are tabled into the tabling area during line activation. The tabling ID used in this cell must be the same as that used in the output completion activity address if output operation is to be done also.

Word 7

This word defines the single input buffer or the input buffer pool. For CTMC/GCS half-word transfers, input characters are transmitted in ascending order within a word starting at the lowest portion of the word. (See 9.4.2 for CTMC/GCS quarter-word transfers.) For both single mode and pool mode, individual buffer sizes are limited to 4095 characters (2047 characters on 524K 1106, 1100/10/20).

input-character-count — Contains the number of characters for single buffer mode; must not be a value that would cause the buffer to extend beyond the upper boundary of the user's D-bank; must contain zero for pool mode.

input-buffer (pool-id (pool mode)) — Contains the single mode input buffer address or the pool-id for pool mode input.

Word 8

For pool mode, this word contains in H1 the end of the queue of buffers presented on each ESI activity activation, H2 may be used by the user to record the next buffer to process in the buffer chain. A1 contains the starting buffer address of the queue of buffers presented on each ESI activity activation. It is the user's responsibility to do any chaining of the present queue of buffers to previously presented and yet unprocessed buffer queues if chaining is desired.

Word 9

partial-buffer-character-count-or-dual-addr — For single mode, H1 contains the number of characters transferred as input when the input transfer of a buffer is completed before the specified count is transferred. For input pool mode, this field must be zero unless dual pool mode is desired; for dual pool mode, H1 contains the dual pool address. Dual pool mode is used primarily for polling operations, whereby a small input buffer can be initially set up so that an immediate switch to a pool of larger buffers occurs when the poll response is received. Thus, larger buffer areas can be used for the input data stream initiated by the polling operation.

buffer-transfer-time — Contains the number of basic time intervals to be used as the maximum time between buffers. If a buffer does not transfer in this time interval, a fault is suspected unless the previous character denoted end of input. If a time value of zero is specified, no timing check is performed by the communications handler (see Word 5).

Word 10

dial status — Contains completion status of the last dial operation. Status codes are:

1_8 - Successful.

2_8 - Unsuccessful. Abandon.

3_8 – Leased line assigned or CTD busy.

4_8 – A hardware parity error occurred during the dial operation or path switch occurred while dialing was active.

5_8 – CTD time out.

6_8 – Unsuccessful. Retry.

10_8 – An ESI activity associated with this line terminal is in a contingency state and as a result the line terminal is terminated.

40_8 – In progress.

dialer–selection status

Contains selected dialer status which is set upon return of the ER CMD$ request. Status codes are:

0_8 – Primary CTD (dialer) is selected.

1_8 – Secondary CTD (dialer) is selected.

3_8 – Both CTDs are busy, dial operation is not initiated.

dial–usage

This parameter denotes the action to be taken upon completion of the dial operation (lower 2 bits) and denotes which dialer to use (upper 4 bits). Values for the lower 2 bits of this field are:

0_8 – No dial completion activity.

1_8 – Give control to dial completion activity when non–ESI Interrupt Tabling dial operation is completed. This value has no effect on ESI Interrupt Tabling dial operation.

2_8 – On CMH$ request, send remote release EF to the CTD only.

3_8 – On CMH$ request, send remote release EF to data CT only.

Values for the upper 4 bits of this field are:

0_8 – Prefer primary dialer.

1_8 – Insist primary dialer.

2_8 – Insist secondary dialer.

dial–completion–activity–address

For non–ESI Interrupt Tabling dial operation, control is passed to the address specified in this field at the completion of the dial operation.

For ESI Interrupt Tabling dial operation, this field contains the interrupt tabling area ID. This ID is the value returned from a CPOOL$ request where the CPOOL$ packet specified a tabling area establishment (word 0, bit 35 = 1). All dial interrupts are tabled into the tabling area

during line activation. The tabling ID used in this cell must be the same as that used in the input and output completion activity addresses if input and output operations are to be done also.

Word 11

This word contains the count of characters in H1, and in H2 the buffer address at which the number to be dialed is stored as decimal digits in successive sixths of a word. The CTMC/GCS dial characters are transmitted in ascending order within a word starting at the lowest portion of the word. The communications handler uses the buffer timer to verify that a connection has been established. This verification is contingent upon the user setting the EOT bit (2^9) in the last dial digit.

9.4. COMMUNICATIONS HANDLER OPERATIONS

9.4.1. Support Operations

The available operations are:

Initialize	ER CMS$ (see 9.4.1.1)
Terminate	ER CMT$ (see 9.4.1.10)
Dial	ER CMD$ (see 9.4.1.2)
Input	ER CMI$ (see 9.4.1.3)
Output	ER CMO$ (see 9.4.1.4)
Input and Output	ER CMSA$ (see 9.4.1.6) (not supported on C/SP)
Hangup	ER CMH$ (see 9.4.1.9)

For C/SP considerations of operations, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version). The references to these operations are made with H2 of register A0 loaded with the address of the LTT defining the LTG.

9.4.1.1. Initialization (CMS$)

Purpose:

To initialize one or more LTGs.

Format:

```
L    A0,(lttcount,lttaddr)
ER   CMS$
```

Parameters:

lttcount — The number of LTTs minus 1 (see 9.3)

lttaddr — The address of the first LTT

Description:

When the CMS$ request is executed, the user program must be in real-time mode, and H1 of register A0 must contain the number of LTGs minus one; H2 must contain the location of the first LTT. The format of register A0 is:

LTT-count-minus-one	first-LTT-addr

where:

LTT-count-minus-one — Specifies the number (minus one) of the contiguously located LTTs which are to be initialized and which must not exceed the number of communications LTGs (minus one) currently assigned to the user.

first-LTT-addr — Specifies the address of the first of the contiguously located LTTs to be initialized

Each LTT must be formatted for the initialize operation. See 9.3 for the format of the LTT.

When a CMS$ request is made, the LTTs specified by the user are initialized, in turn, beginning with the LTT specified in H2 of register A0. Thus, when an error condition is detected and control is returned to the user's error contingency routine, all of the LTTs preceding the one being initialized when the error condition is detected have been initialized. The user may elect to use the initialized LTTs or terminate them by executing a CMT$ request (see 9.4.1.10).

9.4.1.2. Dialing (CMD$)

Purpose:

Initiates a communications handler dialing operation.

Format:

```
L,U  A0,lttaddr
ER   CMD$
```

Description:

The user program must be in real-time mode, and register A0 must contain the location (lttaddr) of the LTT (see 9.3).

The LTT must be formatted for the dial operation (see 9.3).

The dial completion activity operates as a high priority interrupt routine and is allowed a limited time period to perform analysis of the interrupt. When this routine is entered, register A0 contains the location of the appropriate LTT.

The dial operation initiates the buffer specified in word 11 of the LTT addressed by register A0. The telephone number to be dialed must be decimal digits in successive sixth words. After the dial operation is complete one of the following occurs; the user receives control at the dial completion activity address if the line was initialized with a dial completion activity; or if interrupt tabling was specified on initialization, the information is tabled and the optional activity is activated if one is associated with the area. If tabling is not requested, then the user optionally may or may not have requested a completion activity. In either case, the dial status is set in S1 of word 10 of the LTT. If input or output was initiated before dialing, the character timing is not done before either the first character transfer or the dial times out. In the case of initiating output, the output terminal requires an enable from the dial unit, so transfer starts immediately upon dial completion. A dial request on a leased line is given a unique status code and subsequent input or output is honored. The dial completion activity operates as an ESI completion activity.

If an automatic calling unit does not exist for an LTG, the following message is displayed on the operator's console:

```
run-id  DIAL  NUMBER..... line-id Y OR N?
```

The operator must respond with an N or Y to indicate completion.

9.4.1.3. Input (CMI$)

Purpose:

Initiates a communications handler input operation.

Format:

```
L,U  A0,lttaddr
ER   CMI$
```

Description:

When a CMI$ request is made, the user program must be in real-time mode, and register A0 must contain the location (lttaddr) of the LTT (see 9.3). This ER returns control immediately.

Input on line terminals can be initiated in one of the following modes:

- Single buffer
- Pool
- Dual pool

9.4.1.4. Output (CMO$)

Purpose:

Initiates a communications handler output operation.

Format:

```
L,U  A0,lttaddr
ER   CMO$
```

Description:

When a CMO$ request is made, the user program must be in real-time mode, and register A0 must contain the location of the LTT (see 9.3).

Output on line terminals can be initiated in one of the following modes:

- Single buffer
- Pool

9.4.1.5. Send and Acknowledge (CMSA$)

Purpose:

Initiates a communications handler input and output operation.

Format:

```
L,U  A0,lttaddr
ER   CMSA$
```

Description:

When a CMSA$ request is made, the user program must be in real-time mode and register A0 must contain the LTT (see 9.3) address (lttaddr). This request is the exact equivalent of the sequence:

```
L,U  A0,lttaddr
ER   CMI$
ER   CMO$
```

9.4.1.6. Single Buffer Mode for Input/Output Operations

If a request to CMI$ (see 9.4.1.3), CMO$ (see 9.4.1.4), or CMSA$ (see 9.4.1.5) is for single buffer mode, the LTT (see 9.3) must be currently initialized for either operation by setting up the appropriate word of the LTT. Word 7 is set up for input operations; word 3 is set up for output operations. If the LTT is initialized for single buffer mode input, the CMI$ request must be for single buffer mode input. Similarly, if the LTT is initialized for single buffer mode output, the CMO$ request must be for single buffer mode output.

For single buffer mode input, a CMI$ request causes initiation of input according to the input access control word (IACW) in word 7 of the LTT; the IACW points to word 0 of the input buffer.

For single buffer mode output, a CMO$ request causes output according to the output access control word (OACW) in word 3 of the LTT; the OACW points to word 0 of the output buffer.

For the CTMC/GCS, the completion of an I/O operation is indicated by a monitor or external interrupt or a timeout.

After the input interrupt is detected, the input line terminal is turned off. If the line was initialized with interrupt tabling, then the interrupt information is tabled and the optional activity is activated, if one is associated with the tabling area. If the line was not initialized with tabling, then the interrupt status of the input buffer is stored in S1 of word 6 in the LTT. The number of characters accepted as input is stored in H1 of word 9. For synchronous line terminal devices working on a CTMC/GCS, there may be extraneous characters in the buffer following the last data character, if the message received is shorter than the buffer. If the line was initialized with interrupt tabling, the number of input/output characters transferred in single mode are presented in the tabled information.

The completion activity for either input or output operations is activated with register A0 set to the starting address of the LTT controlling the buffer, and with register A1 set to the address of the buffer just transferred. The address of the input completion activity is specified in H2 of word 6 of the LTT; H2 of word 2 of the LTT contains the address of the output completion activity routine. This routine is given control as an ESI completion activity.

H1 of word 9 of the LTT contains the number of input characters before the input completion activity is activated. H1 of word 5 contains the number of characters transferred as output if the output of a buffer is completed before the specified count is transferred.

H2 of word 9 of the LTT contains the number of basic time intervals to be used as the maximum time until the input buffer is filled. For output buffers, H2 of word 5 contains a similar value. The basic time interval used by the system is 600 milliseconds for I/O buffer transfers. If a buffer is not filled in the time interval specified, a fault is suspected unless the previous character indicated end-of-message. The timeout status is either tabled or passed to a completion activity through the LTT, depending whether the initialization of the line specified tabling or completion activity control.

For C/SP considerations regarding single mode, see UNIVAC Communications/ Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

9.4.1.7. Pool Mode for I/O Operations

If a request to CMI$ (see 9.4.1.3), CMO$ (see 9.4.1.4), or CMSA$ (see 9.4.1.5) is for pool mode input or output, respectively, the LTT (see 9.3) must be currently initialized for pool mode input or output operations. All CMI$, CMO$, or CMSA$ requests must be for pool mode, and there must be at least one buffer currently available in the pool.

For input, pool mode is indicated by a zero value in H1 of word 7 of the LTT. H2 of word 7 of the LTT points to a pool control packet which locates the pool of chained buffers. The value specified in H2 of word 7 is the address of the pool control word returned in H2 of register A0 upon return from a CPOOL$ request (see 9.4.2.1).

For output operations, pool mode is indicated by a zero value in H1 of word 3 of the LTT. H2 of word 3 points to a pool control packet which locates the pool of chained buffers. The value specified in H2 of word 3 is the address of the pool control word returned in register A0 upon return from a CPOOL$ request (see 9.4.2.1).

The output queue may contain any number of buffers between one and the total number in the pool. These buffers must be currently removed from the output pool via ER CGET$ and not be currently in another queue.

In the input pool mode, input is initiated with monitor. As each input buffer is filled, it is added to the backup queue specified in H1 of word 8 of the LTT. The user program is expected to update H2 of word 8 with the link portion of each buffer after that buffer has been processed. The user program can determine the end of the input backup queue by updating H2 of word 8 until either a value of zero is encountered or it matches H1 of word 8. The first word of the buffer that is added to the backup queue is loaded with the completion status in S3 of word 0 and the number of input characters transferred in T1 of word 0. Depending upon the input-usage field, S3 of word 6 of the LTT, the input completion activity is activated. A monitor interrupt causes buffer switching. Upon receiving an end-of-message indication, S2 of word 6 of the LTT is tested to decide whether to set up input into another buffer (nonzero value) or to turn off the input operation until the next request by the user program. The user program must add the completed buffers back to the available pool.

In the output pool mode, each buffer is transferred with monitor and, upon interrupt, the next buffer in the backup queue is initiated. As each buffer is removed from the queue, the communications handler updates the start of the output-backup-queue field (H2 of word 4 of the LTT). The user program can dynamically add buffers to the output queue by updating H1 of word 4 of the LTT and issuing an ER CMO$ after having chained the new buffers out of the last old buffer. As each output buffer is emptied, the communications handler stores the completion status in S3 of word 0 of the buffer and the number of characters transferred in T1 of word 0. If a nonzero output status is received, the communications handler returns all the remaining buffers in the output queue to the user with this status in S3 of word 0 of all the buffers and a zero character count in T1 of word 0 in all but the active buffer which contains the actual character count at the time of the interrupt. The end of buffer flag (020_8) is also set in the status word of the last buffer. Depending upon the contents of the output-usage field (S3 of word 2 of the LTT), the output completion activity is activated. The communications handler examines the start of the backup-queue field (H2 of word 4) when the output is first initiated, then works from the link field (H2 of word 0 of the buffer) to determine the end of the chain. When the end of the backup queue is reached, the communications handler turns off the output and returns a status code of 20_8 to denote the caught up condition. If the user program submits new buffers, the output must be reestablished by a CMO$ request (see 9.4.1.4) and the start of the backup queue must be reset.

Sharing buffer pools (as established by CPOOL$) between input and output and between line terminals is fully supported for all Series 1100 Systems. However, to take full advantage of the data chaining feature of the 1110, 1100/40, 1100/80 Systems and to prevent the possibility of data destruction, pools which are shared are used in the active and back-up control word mode with chaining occurring only from active to back-up. Back-up control cell replenishment is done by the communications handler without intervention on the part of real-time programs. This restriction provides an environment in which both nonsharing of pools (full use of hardware data chaining) and sharing of pools (software data chaining) are possible. If a line has been initialized with interrupt tabling, then all pool mode input and output interrupt information is tabled into the tabling area and the optional activity is activated if needed.

Both the input and output completion activities are activated with register A0 set to the starting address of the associated LTT and register A1 containing the address of the buffer or user's data area. The completion activity is given control as a high priority interrupt processing routine and is therefore allowed minimum time for analysis of the interrupt. Completion activity timing considerations are discussed in 9.5.

For C/SP pool mode consideration, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version).

9.4.1.8. Dual Pool Mode for Input Operations

When using dual pool mode for input operations, word 7 and H1 of word 9 of the LTT (see 9.3) are used for dual pool mode input. Dual pool mode for input is specified when H1 of word 7 is equal to zero and H1 of word 9 points to a buffer pool control packet different from the buffer pool control packet specified in H2 of word 7. Input is initiated using the dual pool specified by H1 of word 9. The first monitor interrupt causes pool switching with input reinitiated using the next buffer in the main pool indicated by H2 of word 7 in LTT. On subsequent monitor interrupts, buffers from the main pool continue to be used. When an external interrupt occurs, input is reinitiated using a dual pool buffer if the End-of-Input-Action flag in the IT is set.

Dual pool mode is not supported on the C/SP.

9.4.1.9. Hangup (CMH$)

Purpose:

Initiates a communications handler hangup operation.

Format:

```
L,U  A0,lttaddr
ER   CMH$
```

Description:

When a request is made to CMH$, the user must be in real-time mode, and register A0 must contain the address (lttaddr) of the LTT (see 9.3).

The CMH$ request releases the current remote connection. At the time the CMH$ request is made, the user program must have ensured that output operations are completed, and that any input operations that may occur are of no concern. The CMH$ request disregards the current line activity and issues a remote release to the dial, if one exists, and to the input line terminals. Any further activity after the hangup must be preceded by a CMD$ request (see 9.4.1.2) to activate the terminal.

If no automatic dialing exists for an LTG and the line is not configured for remote release, a CMH$ request displays the following message on the operator's console:

```
run-id HANGUP  line-id
```

No response is required by the operator for this message.

9.4.1.10. Termination (CMT$)

Purpose:

Deactivates the input and output line terminals and performs various housekeeping functions associated with an LTG.

Format:

```
L,U  A0,lttaddr
ER   CMT$
```

Description:

When a request is made to CMT$, the user must be in real-time mode, and register A0 must contain the address (lttaddr) of the LTT (see 9.3).

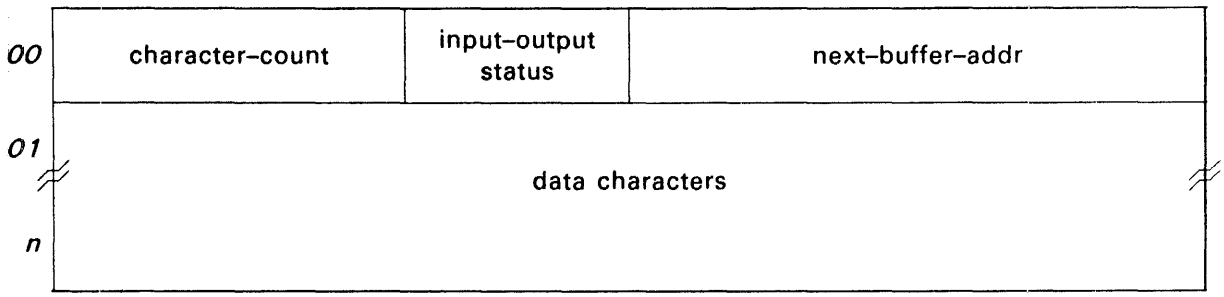
The CMT$ request terminates the line terminal associated with the LTT. The assignment for the device is not released from the user program; it can be reinitialized by CMS$ (see 9.4.1.1). A communications device is released either by an @FREE control statement (see 3.7.4) or when the program terminates.

9.4.2. Communications Pools and Interrupt Tabling Area

A pool for use with the communications handler may be established in any portion of an active user bank. The system furnishes the subroutine to:

- establish a communications pool (CPOOL$ – see 9.4.2.1.1)
- remove buffers from pool (CGET$ – see 9.4.2.2)
- return buffers to pool (CADD$ – see 9.4.2.3)
- expand a communications pool (CJOIN$ – see 9.4.2.4)
- release one pool or all pools (CREL$ – see 9.4.2.5)
- establish a communications interrupt tabling area (CPOOL$ – see 9.4.2.1.2)

The format of each buffer in a pool is:



Word 0

character-count — Specifies the number of characters transferred to or from the buffer. For input buffers to be processed by the input completion activity routine, the character count for completely filled buffers is the value which is specified by the pool start information in H2 of word 8 of the LTT (see 9.3).

For partially filled input buffers, which may occur as a result of either a time out or an external interrupt, this field contains the number of

characters in the buffer when the timeout or external interrupt occurred.

For output buffers, the character count represents the number of characters to be transmitted from this buffer. Character count for output buffers can be dynamically supplied by the user so that he may specify either partially or completely filled buffers in any order.

input-output-status — Contains the status code for this buffer. Status codes in this field are identical to those described for S1 of word 6 (for input), and S1 of word 2 (for output) of the LTT (see 9.3).

next-buffer-addr — Specifies the address of the first word of the next buffer in the chain of those currently linked together. A value of zero in H2 of the last buffer is interpreted as the end of the chain.

Word 1

For C/SP control devices the following additional information word is used in addition to Word 0.

line-spacing	punch-option	ICA format	terminal-id

line-spacing — The number of lines to be spaced before printing data. This function is ignored unless the device is under system control, see UNIVAC Communications/Symbiont Processor System 1100 Series Supplement Programmer Reference, UP-7917 (current version). A value of $0\text{-}7776_8$ skips the specified number of lines before printing. A value of 7777_8 homes the paper on the printer.

punch option — If 0, use translate option if available. If 1, punch column binary.

ICA format — ICA data transfer format field is applied by the user for output and filled by the handler on input. A value of 0 is format A (Fieldata, six characters per word). A value of 1 is format B (ASCII, four characters per word). A value of 2 is format C (binary data bit stream).

terminal-id — This field identifies the particular terminal on a single or multipoint line and the device on the terminal if an auxiliary device interface is configured.

Bits		Value Range
0-7	1-255	Logical station or terminal number (point-to-point station must be 1).
8-11	0-15	Auxiliary device number 0, if no auxiliary interface, 1 is the first device configured, etc.

Words 1 to n for CTMC/GCS

Words 2 to n for C/SP

The value in the character count field determines the value of n.

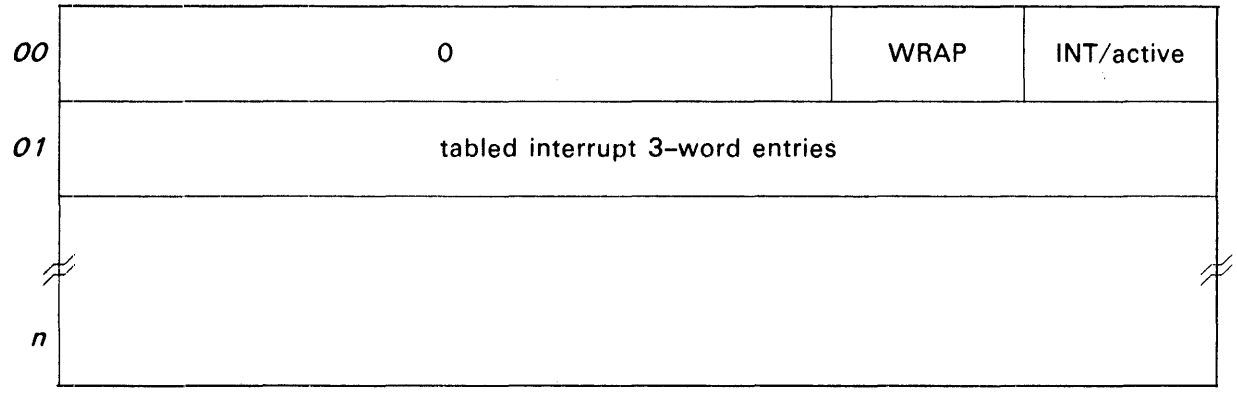
The format for data received or transmitted for CTMC/GCS quarter-word mode is to or from successive quarters of the input or output data as follows:

character-1	character-2	character-3	character-4
character-5	character-6	character-7	character-8
character-(n-3)	character-(n-2)	character-(n-1)	character-(n)

The format for data received or transmitted for CTMC/GCS half-word mode is to or from successive halves of the data area as follows:

character-2	character-1
character-4	character-3
character-n	character-(n-1)

The format of the interrupt tabling area is as follows:



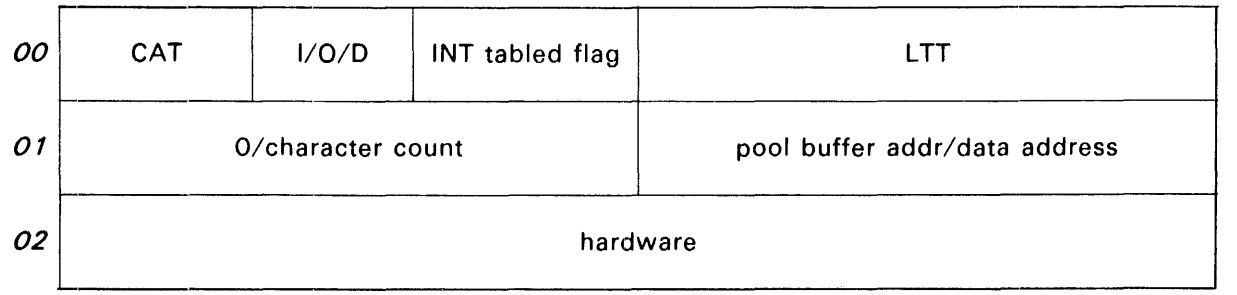
Word 0

WRAP — Indicates to the user that when the communications handler tabled information, it overlayed the first word of the entry with new information and the INT tabled flag had not been cleared. It is the user activity's responsibility to clear the INT tabled flag of each entry on processing the tabled information. The WRAP flag indicates possible loss of interrupt information.

INT/active — This flag indicates to the user that an interrupt tabling has occurred. For tabling areas without an associated activity this flag should be checked whenever the processing activity is activated.

For interrupt tabling areas with an associated activity the flag also indicates to the communications handler that the activity is active and an activation on interrupt occurrence is not needed. This flag should be cleared by the user when it is determined that no further interrupt processing is required. The program should make a test for tabled interrupt after clearing the flag to guarantee that activity activation is performed and an interrupt is not left tabled until a following interrupt occurrence.

The format of the tabled interrupt is:



Word 0

CAT — Interrupt status (see Table 9-1)

I/O/D — 0 – output interrupt
1 – input interrupt
2 – dial interrupt

INT tabled flag	Interrupt has been tabled flag
LTT	Line terminal table address

Word 1

0/character count	0 – pool mode character count – single mode
pool buffer addr/ data address	User buffer or data address

Word 2

hardware	Hardware status presented

9.4.2.1. Establishing a Communications Pool or Interrupt Tabling Area (CPOOL$)

9.4.2.1.1. Establishing a Communications Pool

Purpose:

Establishes a pool of I/O buffers for communications usage.

Format:

```
L,U  A0,pktaddr
ER   CPOOL$
```

Description:

When a CPOOL$ request is made, the user program must be in the real-time mode. The format of the CPOOL$ packet is:

00	mode	char-count	addr-of-first-buffer
01	not used	method	length-of-area-to-be-used

Word 0

mode	Word mode indicator. A value of 0 indicates half-word mode and a value of 1 indicates quarter-word mode. For half-word mode, the user's data area is established in individual buffers equal in length to the character count divided by two, plus one for the buffer control word. For quarter-word mode, the process is similar with the exception that the character count is divided by four. If any divisions yield a remainder, a value of one is added to the quotient. Quarter-word and half-word buffer formats differ only in the total length required to establish the individual buffers.

char-count

The number of characters to be used for each buffer area to be established. The value specified in this field should be the optimum value for the application. The value specified may be either an odd or even number but it must not exceed the maximum communications buffer length defined in the system generation. One main storage location is assigned to the individual buffer area for each group of two characters (half word) or four characters (quarter word).

addr-of-first-buffer

Specifies the starting main storage address of the area to be set up as an input/output buffer pool.

Word 1

method

Defines the method of pool buffering for which the established buffers are to be used. Two methods of pool buffering are permitted.

The first method is referred to as the open chain method. Each individual buffer is linked to the next buffer in the pool by the value in H2 of word 0 of each buffer, except for the last buffer in the chain which has a value of zero in its link field. A zero in the method field causes the pool to be open ended; hence, the name, open chain method.

The second method of pool buffering is referred to as the continuous chain method and is specified by a nonzero value in the method field. Each individual buffer is linked to the next buffer in the pool in the same manner as employed by the open chain method, except the last buffer in the sequence is linked back to the very beginning of the pool, thus forming a continuous chain. Closed pools cannot be shared.

The open chain method is the preferred method because each buffer is removed from the pool by the communications handler as input data is received and is not used again until it has been returned to the pool by the user. In the use of the continuous chain method, an individual buffer is never really removed from the pool but rather the buffers in the pool are used in sequentially cyclic manner. This may result in the reuse of a buffer before all of its previous contents had been processed because either the buffer is of an insufficient size for the application or the real-time program may be spending excessive time in its buffer processing.

One of the more frequent uses of the continuous chain method is to employ two individual buffers chained to each other, thereby operating in an alternating toggling manner. When the continuous chain method is employed, the user assumes all responsibility for processing individual buffer contents in the required time interval. For the preferred open chain method, the communications handler ensures that no buffer is reused until directed by the real-time program. For either method, the optimum size buffer must be used for the application.

length-of-area-to-be-used

Specifies the length of the main storage area to be used for the pool. The setup routine continues to establish individual buffers of the specified size in the desired method until this value is exhausted.

Upon return from the CPOOL$ request, a pool-id is in H2 of register A0. The user is expected to place the pool-id in every LTT (see 9.3) sharing the pool. If the pool is used for output, the pool-id is placed in H2 of word 3 of the LTT. If the pool is used for input, the pool-id is placed in H2 of word 7 of the LTT. A pool may be used for both input and output by placing the pool-id in H2 of register A0 in both word 3 and word 7 of the LTT. Please note that the pool-id returned in H2 of register A0 is an Executive linking value to be used only for Executive control of the buffer pool. The only circumstance under which the real-time program can use this value is in providing it as information for an Executive Request such as CGET$.

9.4.2.1.2. Establishing an Interrupt Tabling Area

Purpose:

Establishes a user data area as an ESI interrupt tabling area.

Format:

```
L,U  A0,pktaddr
ER   CPOOL$
```

Description:

When a CPOOL$ request is made the user program must be in the real-time priority level. The format of the packet is:

00	word length	start of area
01	zero or activity name	

Word 0

Bit 35 — This must be set to a one (1) to indicate that this CPOOL$ request is to establish a tabling area.

Word Length — Contains the length of the area to be used for interrupt tabling. This value must be equal to or greater than four (control word plus one three word interrupt entry) and must be less than 0200000_8. The length need not be exactly the length of interrupt entries plus one. The handler skips the last extra word(s) if there is no room for another entry.

start of area — Address in user write enabled bank where interrupt tabling area begins.

Upon return from the CPOOL$ request a tabling area ID is contained in H2 of register A0. The user should place the ID in sections to be initialized of every LTT to be associated with the tabling area.

If input is to be initialized, the tabling ID should be placed in the input ESI completion activity starting address cell in the LTT (H2 of word 6); and if output is to be initialized, the ID should be placed in the output ESI completion activity address cell (H2 of word 2) in the LTT. On a CMS$ request, the

input and output usage cells in the LTT should be set to 03_8 to indicate that tabling is to be used for interrupts instead of completion activities. All interrupts from a line are tabled if the line is initialized with interrupt tabling. The user should remember that the only ER request that can be performed on a tabling ID is a CREL$. All other (CGET$, CADD$, CJOIN$) are errors and are returned an error code through the user contingency if present. The value returned from a CPOOL$ on a tabling area request is to be used only as LTT information on other ER requests.

9.4.2.2. Removing Buffers from a Pool (CGET$)

Purpose:

Removes buffers from the pool only when the open chain method is employed. Any number of buffers can be removed from the pool for future exclusive use by the requestor.

Format:

```
L    A0,(nbr-buffers, linking-value-for-pool)
ER   CGET$
```

Description:

The program must be in real-time mode at the time of the CGET$ request.

To remove buffers from the pool, register A0 must contain the following:

nbr-of-buffers-to-be-removed	linking-value-for-pool

The return of control from the CGET$ request is with the information provided in register A0. H1 of A0 is the actual number of buffers removed from the specified pool; this value is normally the number of buffers requested unless less than that specified number existed in the pool at the time of the CGET$ request. H2 is the starting address of the buffers removed. Each buffer removed is linked to the others by the open chain method.

9.4.2.3. Returning Buffers to a Pool (CADD$)

Purpose:

Returns a number of buffers to the pool.

Format:

```
L,U  A0,pktaddr
ER   CADD$
```

Description:

The program must be in real-time mode at the time of the CADD$ request and control register A0 is loaded with the address of a packet containing the following information:

00	nbr-of-buffers-returned	linking-value-to-buffer-pool
01		addr-of-first-buffer

A buffer can be returned only to the pool from which it had been previously removed so that buffer size consistency can be maintained within a pool. Each buffer returned to the pool is expected to be linked to the others by the open chain method.

9.4.2.4. Expanding a Pool (CJOIN$)

Purpose:

Expands or adds to a previously established pool by joining it to an additional pool area.

Format:

```
L,U  A0,pktaddr
ER   CJOIN$
```

Description:

H2 of A0 must be loaded with the address of a 2-word packet whose format is:

00	name-of-pool-to-be-expanded	starting-addr-of-added-pool-area
01		length-of-added-pool-area

In addition to loading register A0 with the packet address, the worker program must be in real-time mode and should have an error contingency routine registered with the Executive (see 4.9.2).

9.4.2.5. Releasing Communications Pool(s) and Interrupt Tabling Area(s) (CREL$)

Purpose:

Release buffer pools and interrupt tabling areas established through a CPOOL$ request (see 9.4.2.1.1). Individual pool and tabling areas or all pools and tabling areas associated with a user program may be released.

Format:

Two formats are available for ER CREL$.

To release all pools and interrupt tabling areas:

```
LXI,U   A0,1
ER      CREL$
```

To release one pool or one interrupt tabling area:

```
L       A0,pool-id or interrupt tabling area-ID
ER      CREL$
```

Description:

The buffer pool-id is returned to register A0 on a CPOOL$ request (see 9.4.2.1.1).

The interrupt tabling area-ID is returned in register A0 on a CPOOL$ request (see 9.4.2.1.2).

A pool-id or tabling area-ID need not be given to release all pools and tabling areas.

9.4.3. Altering Communications Paths (ROUTE$)

Purpose:

Dynamically alters the primary paths of communications LTGs.

Format:

```
L     A0,(mode,lttaddr)
L,U   A1,pointer
ER    ROUTE$
```

Parameters:

mode	1_8 – Request alternate input LTG path
	2_8 – Request alternate output LTG path
	3_8 – Request alternate LTG paths for both input and output
lttaddr	The address of the LTT (see 9.3)
pointer	Specifies a logical alternate of the primary LTG, such as 1, 2, 3, ..., n

Description:

Each primary LTG is defined in the system generation and provides the necessary information for the communications handler and facility inventory to make assignments by means of the @ASG control statement (see 3.7.1). The primary LTG may define up to three terminals: input, output, and dial; therefore, an alternate LTG configuration may define one, two, or three terminals.

An alternate LTG path may be another assignable LTG primary path or may be assignable only by the ROUTE$ request. If a primary LTG path is reassigned (altered) to another primary LTG path, both

input and output line terminals should be altered; otherwise, the first primary LTG path remains in an assigned state and cannot be reassigned. Once a primary LTG path has been altered, all dial and hangup operations are initiated using the new LTG path. If, however, the primary LTG path was only partially altered (input or output, but not both), the communications handler provides dialing specified by the output line terminal alternate only.

It is the responsibility of the user to perform manual dialing for an LTG not dialed by the communications handler. If both input and output line terminals of a primary LTG path are altered, the handler performs the hangup operation for the primary LTG path.

9.4.3.1. Routing Procedures

Before a ROUTE$ request can be made to the communications handler, the user's LTT (see 9.3) must be initialized by using the CMS$ request (see 9.4.1.1). Once the LTT has been initialized, the ROUTE$ request may be referenced as frequently as desired. Once a primary LTG has been routed, the primary LTG is not available for this assignment. Either an @FREE control statement (see 3.7.4) must be initiated or the primary LTG may be an alternate of its alternates and the ROUTE$ request may be initiated to reestablish the original assignment. If the primary LTG being routed has an idle line monitor state and both input and output are routed, the primary LTG is reestablished in the idle line monitor state.

9.5. COMPLETION ACTIVITIES

Completion activities, called externally specified index (ESI) activities, are given control at the occurrence of ESI interrupts for CTMC/GCS subsystems or internal specified index (ISI) input interrupts for the C/SP subsystem. Completion activities are initialized by the communications handler when given a CMS$ request (see 9.4.1.1).

The user can request completion activities for input, output, and dialing line terminals by specifying a usage code and completion activity location in the LTT for each respective mode of operation.

When control is given to an ESI activity, register A0 contains the address of the respective LTT (see 9.3); register A1 contains the address of the first word of the user data area for single buffer mode; register A1 contains the address of the user's buffer location for the buffer pool mode. The complete set of A-, X- and R- registers may be used in ESI activities, but their contents are not passed on or restored when the activity releases control. Also, the control registers are not passed on between real-time and ESI activities.

ESI activities are given control as high priority interrupt processing routines. These activities are interrupted only by the real-time clock, the day clock, ESI and ISI interrupts; therefore, it is necessary that these activities should be timed to detect closed-loop situations and other excessive computation. The time quantum is variable and is specified in the system generation. The quantum used should reflect that the ESI activity is interruptable as previously described and should account for the time lost in processing those interrupts. If the ESI activity execution time exceeds the specified amount, the activity is placed in contingency state, as described in 4.9.5.

Interrupt tabling should be used rather than completion activities, whenever possible, as it results in a substantial reduction in system and program overhead. ESI completion activities are provided for the cases where a time critical action is required in order to properly control the line or terminal device.

Programs to which communications devices are assigned utilizes PCT area to describe the completion activity. For every line assigned by the program with the completion activity addresses equal to a previous line a 016 word PCT saving results. The saving in PCT area is most apparent for larger networks.

The following requests may be used to allow an ESI activity to release control:

- ER EXIT$ (see 4.3.2.1)
- ER ACT$ (see 4.3.3.4)
- ER CADD$ (see 9.4.2.3)
- ER ADACT$ (see below)

Reference to any other request causes an ESI contingency condition.

Release of control by an ESI activity should not be confused with activity termination. ESI activities are terminated only by a CMT$ request (see 9.4.1.10) or an @FREE (see 3.7.4) of the LTG.

- Exiting from an ESI Activity (ADACT$)

Purpose:

Used to exit from an ESI activity, return specified buffers, and activate a previously named activity.

Format:

```
L     A1,name
L,U   A0,pktaddr
ER    ADACT$
```

Description:

The name parameter specifies the symbolic name returned in register A0 as a result of a NAME$ request (see 4.3.3.2).

For ADACT$ requests, register A0 must contain the address of a 2-word packet whose format is:

nbr-of-buffers-returned	linking-value-to-parent-buffer-pool
	addr-of-first-buffer

The format of this packet is identical to the format of a 2-word packet used for the CADD$ request (see 9.4.2.3).

9.6. IDLE LINE CONTROL

9.6.1. Idle Line Monitor

In the system generation, the input line terminal devices can be given an unassigned status of either off or standby. The standby status causes input to be enabled on the devices when not assigned to a program. Upon receipt of a particular identifying character string the appropriate symbiont is initiated.

9.6.2. Idle Line Polling

Systems generation can also specify that an idle line be polled by the Communications Handler. This is accomplished by determining which idle lines are to be polled and what poll is to be sent of the configured polls using history cells. The poll message is then sent to the line. Activation is done each six (6) seconds.

9.7. TIMING CONSIDERATIONS

In communications equipment handling there are three levels of activity which must be considered to determine the communications activity which can be allowed in conjunction with other I/O activity, real-time clock activity, and processor usage. These levels are:

- Interrupt response
- Buffer processing
- Information analysis

9.7.1. Interrupt Response

Interrupt response occurs within the communications handler and consists of setting up the next buffer for the interrupting line terminal, queuing the interrupt or the interrupt table, and reinitiating the input or output mode. The necessary criteria to be met by the handler are:

1. Ensuring that the mode be reset within the character availability of the fastest devices on the channel (160 microseconds on 50K baud (7-level plus parity)).

2. Ensuring that each interrupt is processed in a time interval such that all active lines could be ready to interrupt at the same time and no information is lost on any line.

If single buffer mode is employed, then the second case is of no concern for the given line (the interrupt terminates activity on that line) but must be considered for lower priority multiple buffer lines. The configuration should be arranged with the highest speed line terminals which may be used in pool mode in the highest priority interrupt position (lowest ESI channel number and highest priority multiplexer position). To ensure no loss of information for line terminal, higher priority interrupts plus the single interrupt for this line terminal must be handled within the character availability (CA) time of that line.

The count of higher priority interrupts must include:

- one for each ESI external interrupt which can occur in the character availability time;
- one for each half-duplex I/O line terminal pair of higher priority;
- one for each simplex line of higher priority;
- two for each higher priority full-duplex pair; and
- one for the line of concern.

If any buffer of a higher priority line can fill once or more within the character availability time, one must be added for each occurrence. The character availability divided by 40 microseconds should be greater than the number computed. This value takes into account the interrupt processing instructions plus data transfers. For instance, with the configuration of:

- 10 full-duplex UNIVAC 1004 subsystems at 4800 bps (CA = 1.25 milliseconds), and
- 250 half-duplex DCT-500/475 teletypewriters at 100 words per minute (CA = 20 milliseconds).

The interrupt count for the lowest priority UNIVAC 1004 subsystem is 19 (two for each of the nine higher priority full-duplex pairs, plus one for the line of concern), and for the lowest priority teletypewriter is 270, which is less than the limits of 1250/40 = 31 and 20,000/40 = 500, respectively; hence, there would be no loss of information if the entire system was operated simultaneously, in a pool mode (remote UNIVAC 1004 subsystem operation under Executive control is normally a single buffer operation).

9.7.2. Buffer Processing

Buffer processing is a user function performed from the input and output completion activities. The most critical buffer processing routines are expected to be those that handle pool mode line terminals. These routines must operate within the constraint that each buffer must be processed within the time it takes to fill the next buffer in the chain (unless the pool is of sufficient length to contain an entire message for each terminal concerned). Each buffer processing routine must share the CPU so that all routines have a chance to process their corresponding buffers. The need to share the CPU among the processing routines dictates that a time interval must be chosen such that switching from one completion activity to another is based on the real-time clock. This time interval is set by a system generation parameter. The basis for determining this interval follows.

Each buffer must be processed in the time interval (TD) to fill the largest (in terms of time) buffer (Tmax) divided by the number of buffers which can be filled in the maximum interval. A given buffer can fill Tmax/TI times, where TI is the time required to fill a buffer (number of characters times the character transfer rate). Tmax is the largest value of TI in the system. Hence, the time interval, TD, is determined by the formula:

$$TD = \frac{Tmax}{\frac{Tmax}{T1} + \frac{Tmax}{T2} + \frac{Tmax}{T3} + \cdots + \frac{Tmax}{Tn}}$$

$$= \frac{1}{\frac{1}{T1} + \frac{1}{T2} + \frac{1}{T3} + \cdots + \frac{1}{Tn}}$$

$$= \frac{T1T2T3T4...Tn}{T1T3...Tn + T1T2T4...Tn + T2T3...Tn}$$

The upper and lower limits of the time interval are determined by T1 = T2 = T3 = ... Tn = Tmin and T1 = Tmin with all other T1...Tn considerably larger which gives:

$$\frac{Tmin}{n} \leqslant TD < Tmin$$

It is reasonable to assume that all devices are buffered so that the time to fill each is nearly equal; hence, TD = Tmin/n can be used. The value n includes:

- one buffer for each simplex input or output;
- one for each half-duplex pair; and .
- two buffers for each full-duplex pair where only multiple buffer mode lines are considered.

The time interval can now be changed as buffers are lengthened or shortened. In the time interval during which the buffers must be processed, the only time which the completion activity does not have control is during cycles taken for data transfers and during time needed to queue interrupts. In the worst case, all active communications lines and all standard lines may interrupt.

Consider the following:

- TD = 10 milliseconds
- 50 active lines at 2400 bps at 7 bits per character
- 1 FH-432 Magnetic Drum Subsystem (240,000 words per second)
- 1 FASTRAND II mass storage unit channel (25,150 words per second)
- 1 UNISERVO VIIIC Magnetic Tape Subsystem channel (16,000 words per second)

In the TD (time interval) given above, there could be (240+25.15+16) (1000) (0.01) = 2811 data transfers at 0.75 microseconds per transfer = 2.1 milliseconds. For high speed channels, 50 (10ms) (1.5 microseconds)/2.91 ms = 0.25 ms for communications line terminal data transfers and (53 interrupts) (25 microseconds per interrupt) = 1.33 milliseconds for interrupts, which leaves 10-(2.1+0.25+1.33) = 6.32 milliseconds as the minimum time the program can count on for executing instructions.

9.8. INFORMATION ANALYSIS

Information analysis is a level of communications activity which is also a user function. This activity is normally expected to be real-time with a user-determined response time. This is handled by the Executive through the standard dispatching algorithm.

9.9. ERROR CODES FOR LT CONTINGENCIES

Each user program to which communications devices are assigned is expected to have an error handling routine registered with the Executive. This error handling routine notifies the user of system and LTG error conditions which do not apply to the transmission or reception of a particular buffer. The user program is notified in the manner and format as defined under program contingency error mode condition. A list of the possible error codes and their meaning can be found in Appendix C.

9.10. PARITY ERRORS

The input, output, and dial operations on an ESI channel via the CTMC/GCS subsystem are subject to receiving a hardware parity error occurred completing status (see 9.3 on input, output, and dial status). This status is the result of an IOAU/Storage Parity Check interrupt or an IOAU/ACR Parity Check interrupt on the ESI channel for an 1110 or 1100/40 System. This status is also the result of an I/O data, Output Data, or Control Word Parity error on the ESI channel for an 1106/1108 or 1100/10/20 System. For an 1100/80 System, this status is the result of one or more of the following errors; SIOF device check, interface control check, channel control check, channel data check, or program check. These errors do not always affect data involved in the actual I/O transfer. The completion activity processing this status should be prepared to handle the following:

- The data involved may have been altered, may not have been transferred, or may have been only partially transferred.

- The IOAU (1110 or 1100/40) detected the parity error in an internal register while processing I/O for this or another line causing sufficient channel delay to warrant notification of all lines that an interrupt may have been lost.

- Upon receipt of the parity status, the line involved may be down. The status is presented on the next ESI completion the user is to receive, irrespective of what status it would have normally received.

The user program should attempt a retry of the previous I/O. Should the line be down, the down status is presented with this completion. For input, analysis of the received data (if any) with the parity status is possible, with the application determining data reliability.

10. Real-Time Processing

10.1. INTRODUCTION

The Executive provides interfaces that enable real-time programs to appropriately influence the Executive. Assistance is provided for real-time programs in the following areas:

1. Activity registration
2. Activity priority manipulation
3. Program positioning in main storage
4. Lockout protection from simultaneous record access during program execution
5. Real-time Test and Set contingency mechanism
6. Inter- and Intra-program Test and Set queuing
7. Interface with nonstandard peripherals at the hardware level (I/O commands and interrupts)
8. Priority access to peripheral devices
9. Communications handler interface to remote devices

The Executive design is not oriented toward any one particular type of real-time application. Instead, it provides a general environment capable of supporting all types of real-time applications.

10.2. PROGRAM LOCATION

The nature of a real-time program causes its storage requirements to be handled differently than ordinary demand and batch applications. A real-time program is never swapped out of main storage nor relocated within main storage, because it must be accessible at all times. For this reason, the Executive system optimally positions all real-time programs in main storage, if necessary, when they initially acquire real-time status. The use of MCORE$/LCORE$ (see 4.7.1 and 4.7.2, respectively) by real-time programs is especially complex due to the "locking in" of the program. MCORE$/LCORE$ requests change the address available to ESI completion activities and communications buffer pools. A full discussion is beyond the scope of this document. However, a basic rule may be stated: the program should expand to maximum size before it acquires real-time status (see 10.3.3) and after an RT$ contract to the desired size.

10.3. BUFFER OPERATIONS

One of the most important considerations in a typical real-time application is the structure and management of communications buffers. Section 9 provides the details of buffer control interfaces with the communications handler. This paragraph discusses the general aspects of communications buffering in the system environment. Although the system provides both single and pool mode capabilities, the user should employ the method most advantageous for the application. The single mode of operation is the most efficient method, as there is no overhead involved with controls for a pool. Single mode uses less main storage per buffer, because pool mode requires additional storage area in the real-time program and within the Executive for the pool control information. This amounts to three words for each buffer in the pool; one word is maintained with parameter information in the user's area, and two words in the user PCT for 1106/1108, 1100/10/20 Systems or EXPOOL for 1110, 1100/40, 1100/80 Systems are used for linking and queuing purposes by the communications handler. The type of transmission is perhaps the best guide as to which should be used.

10.3.1. Transmission Types

Types of transmission may be classified as fixed length, variable length, or indeterminate length. An example of a fixed-length transmission is the use of a UNIVAC 1004 Processor as the remote terminal or a similar printing device which employs a print line image of 80 or 132 characters. Also the poll message (not to be confused with a poll response) for a polled network is generally fixed length. The single mode of operation should always be used for fixed-length transmissions.

The maximum length of a variable-length message can be predicted. An example of a variable-length message is a transmission for a CRT operating as a remote inquiry and display device where the remote CRT user may enter any quantity of information up to the maximum size of the CRT screen. If the amount of main storage available for use as buffers for variable-length messages is extremely limited, it is better to use a pool of buffers since not all messages are in progress at once. Less buffer area is needed than if single mode is used and buffers are permanently assigned. Either single mode or pool mode could be used for a variable-length transmission, depending on which is the most advantageous method for the application.

Indeterminate-length messages should always be processed using the pool mode of operation. A transmission of indeterminate length is most frequently encountered in message-switching applications where the length of an input message is under the jurisdiction of the remote station, and the system must use segmentation to accept, process, store, and forward the entire message.

10.3.2. Main Storage Availability

Another factor which largely affects whether single or pool mode of buffering is to be used is the quantity of main storage available for buffer areas. The lack of adequate buffering areas dictates that the pool method should be used so that the buffer area may be shared by numerous line terminal (LT) groups. Sufficient buffer area may permit the use of single mode buffering or even the extreme case of a closed pool of buffers for each LT group. However, such an extreme case has an added restraint that each buffer of information must have sufficient staging area and adequate mass storage transfer time so that no data is lost during an overload situation. The open chain pool method ensures that a buffer is never reused by the system until so instructed by the user. If pool mode is chosen, consideration must be given to the size of the main storage area to be used for buffering. If adequate area is available, the desired size can be set aside for the pool.

10.3.3. Pool Size

The optimum pool size is determined by the application, but it is also influenced by the system's work load. If the application is such that the loss of any data cannot be tolerated, there is no choice but to fix the size of the pool at some maximum value to adequately handle the peak load. Systems with such stringent requirements do exist, but the real-time program has some degree of control over the remote stations. For example, the polling operations can be reduced if an excess work load is encountered, or the remote station can be instructed to retransmit if any portion of a transmission is lost just as though a parity error had occurred.

It is possible to dynamically modify the size of the pool area using MCORE$/LCORE$ (see 4.7). If this method is used:

1. The pool size is not modified by ER MCORE$/LCORE$; only the available area is modified.

2. The activity requesting expansion/contraction is deactivated until the request is completed; this may include the time necessary to swap programs.

3. Other activities of the program are deactivated only long enough to update resident Executive tables. ESI completion activities are not affected except for the addressing range available to them.

10.3.4. Buffer Size

The proper size for the individual input and output communications buffer areas (not to be confused with mass storage buffering areas) is very closely connected with the size of the pool. The size of a communications buffer is normally fixed at a single adequate value and is not dynamically changed or influenced by changes in the system work load or time of day. The individual buffer size is again determined by the application, but it is greatly influenced by such things as the mass storage medium, staging area and working area size, actual line speeds of the communications network, number of circuits in the network, and data packing techniques. With all of these factors taken into account, typical buffer sizes for existing real-time installations range from 10 to 100 characters. The mass storage medium is perhaps the largest influence in determining the size for individual buffers. A mass storage device with a relatively long access time and low data transfer rate dictates that a larger buffer size should be used, while smaller buffers may be successfully used if the mass storage device has a relatively short access time and a high data transfer rate. Other considerations for determining buffer sizes include: hardware characteristics such as addressability; software features such as the amount and method of segment and message linkage; and real-time factors such as directory contents and location and repacking principles.

Addressability of the mass storage medium determines staging area size which, in turn, controls the size of individual buffers. It is best to establish a buffer which is an integral fraction of the size of the staging area. For example, a mass storage device with either track or sector addressability would need the staging area to be a multiple of the track or sector size. The full range of software techniques for proper use of mass storage is beyond the scope of this manual. The size of individual buffers must increase with the line speed (a larger buffer size is more successful for a 4800-baud line than the buffer size chosen for a 75-baud line). The size of buffers should also increase with the number of lines controlled by the system. Note the use of the term *system*; it is possible that several real-time programs may operate concurrently. The work load would be divided among various programs, according to type of work, with common linkage between programs existing only in the form of queues, tables, files, and so forth. Multiple real-time programs are not time shared or sliced by the Executive, but are expected to voluntarily share the system resources according to their combined requirements.

Data packing techniques also influence individual buffer size, and cover both hardware characteristics and software methods. Hardware characteristics include such things as half-word or quarter-word communications buffering methods, partial-word addressing capabilities of the CPU, and peripheral subsystem operation such as the reading of bytes by way of the UNISERVO 16/20 Magnetic Tape Subsystems, which pack 9-bit bytes in double-word format.

Software methods include use of a common internal code and related packing. Some internal code must be selected as the standard to be used internally by the real-time program. The code which is most common throughout the communications network is generally selected, although the decision may be influenced by line speed to avoid excessive character translation on high speed lines. This selection is generally not Fieldata (which is used internally in the Series 1100) but either Baudot or ASCII, since the majority of remote stations have hardware design characteristics employing one of these codes.

Any input which is received and which does not conform to the standard internal code is translated before being processed by the real-time program. For output, the data in internal code must be translated to the code desired by the remote station as the communications output buffer area is being filled.

Once a particular code is selected as the internal standard for the real-time program, various data packing methods by the software can be built around that code. For instance, the selection of ASCII permits only nine characters to be packed into a double word, while the use of Baudot allows 10 characters to be packed into a double word with the added advantage of two bits remaining for use as control information. The proper control techniques and the greater efficiency of packing using a smaller code such as Baudot can provide a larger system capacity when mass storage area is limited. The use of vertical packing rather than horizontal packing can eliminate the output staging area since the communications output buffer area can double as the staging area. Vertical packing may also be advantageous for certain unique hardware characteristics.

10.3.5. Dual Pool Method

The dual pool method is available for accepting input. The primary use for the dual pool input mode is to provide a rapid software response for features otherwise provided by hardware options for polling operations. The response from a poll message may be either a short response or a lengthy transmission of data from that particular remote station. It is most desirable for the real-time program to have immediate notification when the first portion of the poll response is received so that the next poll message may be initiated if the response indicates NO BUSINESS. Rather than use a timer, the occurrence of an input monitor interrupt for a small buffer can be used to trigger the real-time program's analysis of the poll response. However, if a lengthy transmission was initiated by the poll message, it is not advisable from a system standpoint to continue with small buffers for accepting input data. The dual pool input mode provides the ability to accept a transmission into a small buffer area when the input request (CMI$ request see 9.4.1.3) is initiated, with an immediate switch to a pool of larger buffers for subsequent portion of the data transmission, if any should follow.

10.4. PROGRAM EXECUTION CONSIDERATIONS

The following discussions cover the areas of priorities, priority control, timing constraints, and the use of the Test and Set (TS) instruction to protect common data areas.

10.4.1. Priority Categories

10.4.1.1. I/O Priority

The Executive assigns an I/O request into one of three categories, depending upon the nature of the activity which submits the request:

1. Real-Time
2. Executive
3. Demand/Batch

For each subsystem, all requests in a priority category are completed before any request for the next lower priority is honored.

Look-ahead techniques are used within a priority category, when appropriate, so that average time for I/O operations may be reduced. Look-ahead techniques may cause I/O requests to be completed in a sequence different from the order of submission; therefore, the program must provide its own protective measures and take the appropriate action if it is concerned with the completion sequence of I/O requests.

10.4.1.2. Dispatching Priority

The Executive provides several methods to be used by real-time programs for changing CPU priority (switching) levels, registering activities, and dispersing the work load among the processors in a multiprocessor environment. The proper use of these Executive functions should be understood so that they are not abused, but are properly utilized with minimum system overhead to achieve a desired goal. This discussion is concerned with the priority control Executive requests RT$, NRT$, FORK$, EXIT$, and UNLCK$, as well as the priority levels imposed on all activities by the Executive. The numerous priority levels should be understood in order to determine the proper use of the previously mentioned Executive requests. Deadline and demand activities have a priority below that of real time and are considered to be grouped with batch for the purposes of this discussion. The main categories of CPU priorities have the following order:

Category	Description
A	All interrupt queuing at the time of interrupt occurrence
B	Any ESI completion activity in order of interrupt occurrence
C	All high priority Executive action
D	Any real-time interrupt activity in order of occurrence (level 1)
E	Real-time activity levels 2 through 35

F All low priority Executive actions

G Transaction program activities

H All demand/batch activities

The occurrence of any work in a higher priority category causes the lower category to be suspended until all higher priority work has been completed. All categories except A are controlled by the software and are, therefore, interruptable and subject to suspension.

- Category A

  Interrupt queuing is the process performed at the occurrence of a hardware interrupt and requires approximately 45 instructions of instruction execution time for an ESI interrupt (see 10.6.2.1).

- Category B

  Category B has the highest software imposed priority and is used for the processing of ESI completion activities.

  In order to detect program errors and excessive loops, the Executive always times ESI completion activities. The amount of time permitted for these activities is fixed at system generation.

- Category C

  This category is used for all high priority operations that the Executive must perform, such as I/O control, interrupt post processing, dispatching, and clock control.

- Category D

  This category is used for real-time I/O interrupt activities at level 1 (see Section 6). This category is always activated with a limited register set (X8-X11,A0-A5, R1-R3).

- Category E

  This category is used for real-time priority levels 2 (highest) through 35 (lowest). All activities at a particular real-time level are serviced before any service is given to lower level activities. There is a maximum of 34 real-time levels available for use by all real-time programs. The user is expected to assign the real-time levels appropriately for the application. Either a limited set or a full set of registers may be used for real-time levels 2 through 35.

- Category F

  This category is used for low priority Executive processing such as storage management, function control, symbiont processing, and so forth.

- Category G

  This category is used for all Transaction Processing activities.

- Category H

  Batch activities are normally subject to suspension by the Executive for purposes of swapping. Batch activities of a program having real-time status (see 10.4.2.1), however, are never

suspended for swapping. This is the only preference given to batch activities of a real-time program. All in-main-storage programs are treated equally as far as batch activity switching is concerned. This means that background batch programs which do a lot of computation might delay the completion of batch activity execution within a real-time program, especially if such activities also perform extensive computation. I/O-oriented batch activities are treated preferentially (see 12.5.6.2 for a full discussion of the switching algorithm).

The implication of the preceding discussion is that care must be taken in assigning time-critical work to batch activities within a real-time program.

10.4.2. Priority Control

The following Executive requests can be used by the real-time program to control the priorities and to distribute the work load in a multiprocessor environment:

- ER RT$ (see 4.3.5.1)
- ER NRT$ (see 4.3.5.2)
- ER FORK$ (see 4.3.1.1)
- ER UNLCK$ (see 6.3.9)
- ER EXIT$ (see 4.3.2.1)

10.4.2.1. Changing Activity Priorities (RT$ and NRT$)

The RT$ request (see 4.3.5.1) is provided for changing priority levels within the real-time program. It can accomplish either of the following:

- change of priority level for a real-time activity; or
- allow a program to acquire real-time status

The RT$ request may be executed by any activity: batch, demand, or real time at levels 2 through 35. A check is performed by the Executive to determine if the run's account number permits real-time activities and the real-time level specified is allowed for the account number.

Closely associated with the RT$ request is the NRT$ request (see 4.3.5.2) which allows a real-time activity to return to its original status (that is, batch or demand). The level for a nonreal-time activity is determined by the Executive and cannot be specified by the user. A program has real-time status whenever it has one or more real-time activities.

10.4.2.2. Application of Multiprogramming to Real-Time

Although the multiprogramming concept (use of multiple activities) is applicable to many types of processing, certain aspects are particularly pertinent to real-time programs. Some important real-time applications of multiple activities are:

- Distribution of the work load among several CPUs (multiprocessing)
- Use of real-time interrupt activities to ensure top priority for turnaround of critical I/O operations

- Registering multiple activities at varying real-time priority levels to perform tasks of varying criticality. (See FORK$ and TFORK$ requests – 4.3.1.1 and 4.3.1.2, respectively.)

- Registering batch activities to perform noncritical tasks in the background

While the above techniques are useful and powerful, the real-time programmer must take care not to overuse the activity concept. Important points to consider include:

- Dispatching overhead. The time required to switch from one activity to another is not negligible and can become excessive if the work load is distributed among too many activities. In many cases, a single activity can perform several tasks in serial fashion more efficiently than several activities performing single tasks in parallel (the RT$ request – see 4.3.5.1 – can be used to adjust activity priority according to the priority of the task to be done).

- Forking overhead. Each new activity requires both time and space for registration. Time is also required for activity termination. Again, a single activity may do as well as several in certain cases. Note that activity control space is taken from the PCT and thus unlimited forking is not possible.

- Register set. Extra time and space are required for saving and restoring the major register set. Therefore, heavily executed activities should use the minor set if feasible. Note that an activity cannot change its register set; however, a FORK$ request followed by an EXIT$ request has the effect of changing set. Note that activity name and id are not retained in this sequence.

10.4.2.3. Interrupt Activity Priority Reduction (UNLCK$)

An interrupt activity is initiated, with a limited register set, at a priority level above all other activities for that type of program. If no other Executive Request is to be used by this activity, an UNLCK$ request should be added to return the activity to the real-time priority level of the activity that issued the I/O request (see 6.3.9).

10.4.2.4. Activity Termination (EXIT$)

Normally the EXIT$ request (see 4.3.2.1) causes termination and deletion of an activity. The EXIT$ request works differently, however, when executed by an activity that was originally initiated as an ESI completion activity. Rather than the activity being deleted, it is placed in a wait condition so that it is again eligible for execution as an ESI completion activity when a communications interrupt occurs. When the activity is in a wait condition, it can be given control almost immediately since the area used to hold information related to the activity is permanently maintained for the ESI completion activity.

10.4.2.5. Timed Wait Considerations

The real-time programmer using timed wait requests (TFORK$ – see 4.3.1.2; TWAIT$ – see 4.3.6) should understand that the wait time for real-time activities is an elapsed time by the clock, during which the activity is suspended. When the specified time has elapsed, the activity becomes available for execution at its priority level. This means that execution does not actually resume until there are no higher (or possibly equal) priority activities requiring CPU time.

Timed waits do require time to process. A large number of activities in short wait loops may effectively lock out lower priority activities.

10.4.2.6. Console Interrupt Handling

Real-time programs which accept unsolicited console input usually benefit from using II$ requests (see 4.6.2) rather than the II contingency (see 4.9). The reason is that the II$ request allows a particular activity to get control at a priority which can be programmer specified; whereas, for II contingency, the Executive may select any activity to process the interrupt (this might be a low-priority real-time or a batch activity which could be delayed for an undesirable period by high priority execution). In addition, the II$ request receives some input, which may eliminate the need for a type and read sequence to find out what the operator wants.

10.4.3. Test and Set Usage

The Test and Set (TS) instruction is available in all systems to protect common data in a user program. For batch and demand activities, the Executive automatically resolves conflicts that may occur in a user program because of activities at different switching priority levels. It automatically degrades by one level any activity that experiences a TS failure (TS interrupt). After level degradation, the Executive forces the activity to the end of the list of activities operating at that level.

For real-time programs, the Executive does not change the activity switching level on the TS interrupt, except for interrupt activities which are dropped to the level of the activity that issued the I/O request. This means that for TS purposes, interrupt activities have an implied level equal to that of the issuing activity. Instead, the user activity is directed to a contingency routine if one is registered (see 4.9). If a contingency routine is not registered, the activity is placed at the end of the list of activities operating at the activity's current level. It is obvious that the real-time program can hang the system if the situation does not automatically resolve itself at the original switching levels, or if the proper action is not taken in the contingency routine to change the switching level in relation to other activities that reference the data. The contingency routine enables the activity to change its level (by an RT$) if a potential lockup exists, or to perform lower priority work for some period of time. Note that a contingency notification on a TS interrupt does not occur unless the activity specifically requests such action, independent of other types of contingencies.

A real-time program may have nonreal-time activities. For activities other than real-time, the TS contingency feature is not available. The real-time program may be delayed indefinitely if activities both real-time and nonreal-time attempt to reference common data. The nonreal-time activity should cause itself to be raised to real-time status before executing a TS protecting the common data; the activity can revert to its original status after the need for the common data reference has ended.

A real-time activity should never have a TS cell set when calling an ER. Although this procedure may work in some cases, it is highly vulnerable to lockup (since ERs commonly require Executive processing that runs at a CPU priority level below real-time), and is strongly discouraged. Furthermore, this practice violates a basic system philosophy concerning TS usage. TSs are to be used to protect short critical sequences and not as a long term logical interlock.

An alternative method of handling Test and Set failures which allows use of Test and Sets between different levels of real-time activities, between real-time and batch or demand activities, and across ERs, is provided by the Test and Set Queuing mechanism (see 4.3.4).

It is illegal for ESI activities to reference common data that is also referenced under a TS condition by real-time activities. No action other than a return of control is taken by the Executive when a TS interrupt occurs in an ESI activity.

10.5. PROGRAMMER'S GENERAL RESPONSIBILITIES

The programmer, prior to interfacing with the communications handler, must:

1. Use options on the @RUN control statement to assure sufficient PCT size (see 3.4.1). (All control packets are built and maintained by the handler in the user's PCT.) Once the program becomes real-time, it is locked in main storage and incapable of being moved for dynamic expansion of the PCT. Note that PCT space is also required for activities created by a FORK$ request (see 4.3.1.1).

2. Assign the line terminal group (LTG) to be controlled by an @ASG control statement (see 3.7.1) or dynamically using a CSF$ request (see 4.10.1.1). The assignment must be made using the arbitrary device format.

3. Declare the program as real-time using an RT$ request (see 4.3.5.1).

4. Prepare the LTT (see 9.3) for each LTG, and the input and output buffers to be used for the data transfers.

The user also should register, by an IALL$ request (see 4.9.3), a program contingency routine as well as an ESI contingency routine. By doing this, the user can obtain error information should an error occur during the interface with the communications handler.

The communications handler/user interface, therefore, should include:

- One or more real-time activities which request, by means of the LTT and ERs, the communications handler to perform input and output.

- ESI completion activities which are given control by the communications handler when the data transfer is accomplished.

- A program contingency routine which is given control by the communications handler when invalid requests are made to the communications handler.

- An ESI contingency routine which is given control by the communications handler when errors are detected during the execution of an ESI completion activity.

10.6. ESI CONSIDERATIONS

10.6.1. ESI Activity Concept

An ESI activity is generated when the real-time activity makes a reference to CMS$ (see 9.4.1.1). A non-real-time activity is not permitted to make this reference. At this time, the Executive establishes an entry point at which control is given upon occurrence of an interrupt for the line associated with the LT group. The ESI activity is never suspended, either voluntarily or involuntarily; hence, no control register or CPU state preservation is required for ESI activities. An ESI activity is given top priority by the Executive such that it is never interrupted except for interrupt queuing. Executive operation can be interrupted at any time to allow ESI processing. The philosophy adopted for ESI activities is that of extending system interrupt processing to the user. For this reason, the amount of work done by an ESI activity is restricted to a minimum, allowing a non-ESI activity to do the remaining work. This implies certain restrictions on programs using the communications handler:

1. Only real-time activities can establish ESI activities.

2. ESI activities are limited to four Executive Requests (EXIT$ - see 4.3.2.1; ACT$ - see 4.3.3.4; CADD$ - see 9.4.2.3; and ADACT$ - see 9.5) and may only use one request per activation. Violation causes an ESI contingency.

3. ESI activities must be completed within a limited time frame defined at system generation.

4. TS protected common data can exist among ESI activities but not between ESI and real-time activities. In other words, a real-time activity that blocks the path of an ESI activity is in error.

Since the ESI activity is not allowed forking requests, any real-time activity which is to process incoming data must already have been established with a "name" (see 4.3.3.2). This name is then used by the ESI activity to initiate processing. When processing is complete, the real-time activity must execute a DACT$ request (see 4.3.3.3). DACT$ differs from EXIT$ in that the activity is not terminated but rather put into an inactive state.

In general, it is more efficient to use interrupt tabling (see 9.4.2.1.2) than ESI completion activities. ESI completion activities should be used only when immediate action to control the line or terminal is necessary.

10.6.2. ESI Timing

ESI interrupt processing and switching times under various conditions have been calculated for real-time programs by counting the actual instructions involved. These counts must be considered as estimates because coding is subject to modification.

10.6.2.1. ESI Interrupts

When an ESI interrupt occurs, various functions must be accomplished before interrupts can be enabled. The amount of time required before interrupts are allowed depends upon the type of interrupt (input monitor, output monitor, or external) as well as the type of operation (single or pool mode). A maximum of 45 instructions is required before interrupts are enabled.

1. If an ESI activity has been interrupted, control is returned to the activity within the 45-instruction limit.

2. If control is to be given to a new ESI activity, approximately 120–130 instructions are required between the time the ESI interrupt occurred and the new activity receives control. The time required depends upon the number of control registers to be saved if an activity has been interrupted, as well as the amount of post-processing required for the activity in the communications handler.

10.6.2.2. Real-Time Activities

If a real-time activity has been interrupted to process an interrupt, approximately 230 instructions are executed between the interrupt and the time real-time activity receives control (the actual time depends upon the type of interrupt being processed). If a new I/O function is available for initiation before the real-time activity receives control, approximately 800 instruction executions are required between interrupt occurrence and return of control to the activity.

11. Checkpoint/Restart

11.1. INTRODUCTION

11.1.1. Uses of Checkpoint/Restart

Checkpoint/Restart is an Executive service which saves a complete or partial copy of a user program (checkpoint), to be restored and continued (restarted) at a later time.

11.1.1.1. Insurance in Case of System Failure

A particularly lengthy program may find it worthwhile to checkpoint itself at intervals, so that if the system fails before the program completes, the program may be restarted from the last checkpoint.

11.1.1.2. Spreading Executions of Long Jobs over Several Computing Sessions

A program may be checkpointed prior to the termination of a computing session, thus allowing execution of particularly long programs to be completed in more than one computing session.

11.1.1.3. Insurance In Case of Program Failure

If a program fails due to bad input data (e.g., bad cards, tapes, files), it may be possible to correct the error and restart.

11.1.2. User Interface

A full or partial checkpoint may be made to tape or mass storage. The "S" option indicates whether a partial checkpoint or restart is being done.

11.2. CHECKPOINT

A checkpoint may be taken at any time. If a partial checkpoint is taken when the run is in control mode, (e.g., @CKPT,S file.element) a null checkpoint is created. A warning message is printed.

11.2.1. Control Statement (@CKPT)

@CKPT,*options file.element,,equip-type,reel-nbr/reel-nbr/.../reel-nbr*

options	A – do not error terminate B – don't save the absolute element being executed. N – don't print diagnostics P – error messages are displayed on the console R – save the PRINT$ file S – only do partial checkpoint T – terminate current run (execution if partial)
file	external filename (optional if reel-nbrs are given). A filename not followed by a period is treated as a file only if the file is assigned and the run is in batch mode. Otherwise, it is treated as an element in TPF$.
element	if an element name is present, the checkpoint element is written to the file and element specified.
equip-type	equipment type if tape file and filename is not cataloged or assigned (optional)
reel-nbr	reel number of the file to which the checkpoint is written (optional)

If an element name is not specified, and the file is not a tape file, the checkpoint is written to the file starting with the first sector.

11.2.2. Executive Request

This CSF$ request is used to create a checkpoint during execution of a program. The format of the character string is the same as the control statement.

11.3. RESTART

A run may be restarted in one of three ways.

11.3.1. Control Statement (@RSTRT)

@RSTRT,*/options run-id,acct-id/user-id,file.element,ckpt-nbr,equip-type, reel-nbr/reel-nbr/.../reel-nbr*

options	A – don't error terminate B – don't use the saved absolute element unless it cannot be found in the original program file. If the B option is omitted the saved absolute element is used. C – use read rather than Forward Space File when looking for the checkpoint J – to assign the input tape when the tape is not labeled N – don't print diagnostics P – error messages are displayed on the console R – @SYM saved PRINT$ file S – perform partial restart only

run-id	6-character run-id
acct-id	account number (optional)
user-id	user-id (optional)
file	external filename. A filename not followed by a period is treated as an element in TPF$.
element	If an element name is present, the checkpoint is retrieved from in the file and element specified.
ckpt-nbr	specifies the checkpoint number at which the restart should occur. (used only for restarts from tape) (optional)
equip-type	equipment type if tape and filename is not cataloged or assigned (optional)
reel-nbr	reel number of tape which contains the checkpoint. (optional)

The control statement is used to create a new run that starts at the point the checkpoint has been taken.

11.3.2. Executive Request

This CSF$ request is used to create runs that initialize the restart of checkpoints. The format of the character string is the same as the control statement.

11.3.3. Initializing Restarts

@RIP,*options file.element,ckpt-nbr,equip-type,reel-nbr/reel-nbr/.../reel-nbr*

options

- A – don't error terminate
- B – don't use the saved absolute element unless it cannot be found in the original program file. If the B option is omitted the saved absolute element is used.
- C – use read rather than Forward Space File when looking for the checkpoint
- J – to assign the input tape when the tape is not labeled
- N – don't print diagnostics
- P – error messages are displayed on the console
- R – @SYM saved PRINT$ file
- S – perform partial restart only

file	external filename. A filename not followed by a period is treated as an element in TPF$.
element	If an element name is present, the checkpoint is retrieved from in the file and element specified.
ckpt-nbr	specifies the checkpoint number at which the restart should occur. (Used only for restarts from tape; optional.)

equip-type — equipment type if tape file and filename is not cataloged or assigned (optional).

reel-nbr — reel number of tape which contains the checkpoint (optional).

This control statement is used to recreate the environment saved at the checkpoint. This control statement is generated for the user by the @RSTRT control statement.

11.3.4. Restart Contingencies

A restart contingency routine may be specified at the program level by the user before the program requests either a full or partial checkpoint. Any user activity may also register a restart contingency routine. These routines receive control upon activation at restart, and may do whatever is necessary before giving control to the normal return. The request is registered using ER IALL$ or CREG$ (see 4.9.3). When the contingency routine receives control, the packet is set up as follows:

		contingency type	return address
	partial flag	checkpoint source	checkpoint number

where:

contingency type — is 6 for Restart

return address — is the address to which control should be returned upon completion of the contingency routine

partial flag — specifies the type of checkpoint

- 0 – full checkpoint/restart
- 1 – partial checkpoint/restart
- 2 – partial restart of a full checkpoint

checkpoint source — specifies which source initiated the checkpoint

- 1 – unsolicited console keyin (CK)
- 2 – Executive Request (CSF$)
- 3 – control card (@CKPT)

checkpoint number — is the checkpoint number which is being restarted

After the contingency routine is completed, the user should return control to the address placed in the packet. This address is the correct address to resume execution, so no adjustment of it is necessary.

It must be noted that if more than one restart routine is registered, they are treated as multiple (non-nested) contingencies (see 4.9.4.4).

The following algorithm is used to determine which activities, if any, are diverted to process a restart contingency.

1. If any activities have the restart contingency registered at the activity application, then:

   a. the program application restart contingency, if any, is ignored;

   b. each activity that is registered for restart is diverted to its contingency routine;

   c. activities without the restart contingency return to their respective reentry points.

or

2. If the restart contingency is registered at the program application, then:

   a. if no activity has a valid address for the contingency packet, then:

      1) the restart contingency is ignored;

      2) all activities return to their respective reentry points.

   or

   b. if at least one activity has a valid address for the contingency packet, then:

      1) if the activity that initiated the checkpoint (via CSF$) has a valid address, it is diverted to process the restart contingency and all other activities return to their respective reentry points;

      or

      2) the first activity encountered that has a valid address is diverted to process the restart contingency and all other activities return to their respective reentry points.

NOTE:

In case 2a above, no restart contingency is processed, even though one is registered. This can be overcome by having the bank containing the contingency packet based at checkpoint time.

11.3.5. File Handling at Restart

Table 11-1. Cataloged File Handling

File State at Restart	File State at Checkpoint	
	Cataloged	To Be Cataloged
Cataloged Assigned	use assigned file	use assigned file *WARNING PREVIOUSLY CATALOGED
Cataloged Not Assigned	assign the cataloged file	assign the cataloged file *WARNING PREVIOUSLY CATALOGED
To Be Cataloged	use assigned file	use assigned file
Not Cataloged Not Assigned	recatalog the file assign the file	assign the file to be cataloged

Table 11-2. Cataloged Mass Storage Reloading

File Options at Restart	File Options at Checkpoint		
	B	M	none
B	Reload	Reload	No Reload
M	Reload	Reload	No Reload
none	No Reload *warning CANNOT BE RELOADED	No Reload *warning CANNOT BE RELOADED	No Reload

11.4. ERROR MESSAGES AND ERROR CODES

Table 11-3. Checkpoint/Restart Error Codes

Error Code (Octal)	Description
* 1	CSF$ reject
* 2	CKRS$ error
* 3	Unrecoverable contingency
* 4	Unrecoverable I/O error
* 5	No INFOR or error processing INFOR
* 6	Cannot assign scratch mass storage file for tape checkpoint
* 7	Executing program element not found, or is MAP level 23 or earlier, or its program file is not assigned or cannot be restored
* 010	Write-enabled or guaranteed entry point common banks attached
* 011	Checkpoint file error: PFP error
012	Data received when expecting end-of-file
* 013	Improperly formatted checkpoint
* 014	Inconsistent controlling information in the checkpoint
* 015	Miscellaneous Executive Request (ER) error
016	Checkpoint/Restart not configured
017	Control mode checkpoint in demand mode
020	CKPAR common bank not present in system
021	Run is using write-enabled common banks
022	Error getting buffer
023	PCT has overflowed
024	Run has ESI activities
025	An activity is using test and set queuing
026	Checkpoint or Restart in progress
0101	Facility reject on ER CSF$ request

Table 11-3. Checkpoint/Restart Error Codes (continued)

Error Code (Octal)	Description
* 0102	Error status returned by ER CKRS$
* 0103	Contingency
0104	Error status returned by ER INFO$
0105	Error status returned by ER MSCON$
0106	Error status returned by ER TLBL$
* 0107	Error status returned by ER IOW$ reading the checkpoint
* 0110	Error status returned by ER IOW$ writing the checkpoint
0111	Error status returned by ER IOW$ reading a file to be saved
0112	Error status returned by ER IOW$ writing a file to be restored
0113	Error in compatibility between operating system and common bank
* 0114	Checkpoint block too long for any type of checkpoint block
0115	Checkpoint block too long for this type of block
0116	Checkpoint block too short for this type of block
* 0117	Checkpoint block identifier out of sequence or not recognizable
* 0120	Checkpoint by privileged run
0121	Tape loss of position on file to be saved

* Errors that cannot be ignored by including the A option.

Table 11-4. Checkpoint/Restart Messages

Associated Error Code (Octal)	Message/Description
3,4	Contingencies: *yyyy* TYPE *xx* CODE *xx* CONT *xx* REENT ADR: *xxxxxx* BDI: *xxxxxx* *yyyy* is a symbolic identifier. The possible values are: I/O, SYMB, IOPR, IGDM, IFOF, IFUF, IDOF, IRST, IABT, IINT, ITS, ERR$, BRKPT, PCTOF, ATC, ER, CONS, COM2, COMM, REP. All *x* are octal digits. If *yyyy* is IGDM, one of the following lines may also be printed: PRIVILEGED INSTRUCTION VIOLATION STORAGE LIMITS/WRITE PROTECT VIOLATION CONTROL REGISTER VIOLATION INTERRUPT LOCKOUT VIOLATION E-BIT VIOLATION TABLE LENGTH VIOLATION If *yyyy* is I/O, the following lines may also be printed: PACKET ADR: *xxxxxx* INT FILENAME: *yyyyyyyyyyyy* PKT+3,T1: *xxx* *zzzz* FILENAME: *qualifier*filename(f-cycle)* where *zzzz* is TAPE or DISC. A SNAP$ of the user's data block may follow the above messages with the snap-id DBANK.
014	RESTART: INCONSISTENT CONTROL INFORMATION IN THE CHECKPOINT
2	RESTART: PARTIAL PCT SAVE ERROR
014	RESTART: CANNOT START DIFFERENT LEVEL CHECKPOINT
6	RESTART: ABSOLUTE ELEMENT IS NOT A CHECKPOINT
013	RESTART: IMPROPERLY FORMATTED CHECKPOINT
7	RESTART: ORIGINAL ABSOLUTE ELEMENT NOT RESTORED
012	RESTART: READ DATA INSTEAD OF END-OF-FILE
2	RESTART: TURN ON ERROR. PACKET TOO SMALL
2	RESTART: TURN ON ERROR. INVALID MAIN-I BDI
2	RESTART: ACTIVITY CANNOT BE RENAMED
3	RESTART: CONTINGENCY DURING ACTIVITY TURN ON

Table 11-4. Checkpoint/Restart Messages (continued)

Associated Error Code (Octal)	Message/Description
	RESTART ABORTED. STATUS IS: *xxxxxxxxxxxx* (2 lines)
1	CSF$ RETURNED STATUS OF *xxxxxxxxxxxx* FROM: @image (@image on second line)
7	PROGRAM FILE PACKAGE ERROR STATUS *xx* PKT ADDR: *xxxxxx* is followed by one of the following lines: NO FIND I/O ERROR OR FILE NOT ASSIGNED PROGRAM FILE NOT DEFINED AS PROGRAM FILE PROGRAM FILE OVERFLOW PACKET ADDRESS OUTSIDE LIMITS
5	CHECKPOINT: EQUIPMENT TYPE OR REELS SPECIFIED FOR MASS STORAGE FILE
5	CHECKPOINT: FILENAME IS INVALID
5	CHECKPOINT: EQUIPMENT TYPE IS INVALID (MUST BE TAPE)
2	CHECKPOINT: INFOR TOO LARGE FOR BUFFER
5	CHECKPOINT: ELEMENT NAME SPECIFIED ALONG WITH EQUIPMENT TYPE AND/OR REEL NUMBERS
6	CHECKPOINT: TEMPORARY FILE CANNOT BE ASSIGNED
7	CHECKPOINT: CANNOT BE TAKEN ON MAP LEVEL 23 OR EARLIER
7	CHECKPOINT: PROGRAM ABSOLUTE ELEMENT CANNOT BE FOUND
5	CHECKPOINT: ELEMENT MAY ONLY BE SPECIFIED FOR CATALOGUED OR ASSIGNED FILE
2	CHECKPOINT: PARTIAL PCT SAVE ERROR
4	CHECKPOINT: CANNOT LOAD BANK FOR I/O OPERATION
5	CHECKPOINT: ELEMENT NAME MAY NOT BE SPECIFIED FOR TAPE FILE
5	CHECKPOINT: RUN-ID FIELD INCORRECT
010	CHECKPOINT: WRITE-ENABLED OR GUARANTEED ENTRY POINT COMMON BANKS BASED
3	CHECKPOINT: ILLEGAL LIJ TO COMMON BANK
WARNING	CHECKPOINT: B-OPTION SET, ORIGINAL ABSOLUTE ELEMENT NOT SAVED

Table 11-4. Checkpoint/Restart Messages (continued)

Associated Error Code (Octal)	Message/Description
WARNING	CHECKPOINT: EQUIPMENT TYPE AND/OR REEL NUMBERS IGNORED FOR ASSIGNED FILE
WARNING	CHECKPOINT: CYCLE IS IGNORED FOR CHECKPOINT ABSLOLUTE ELEMENT
0101	*block-id CSF$ image* *CSF$ message* *CSF$ message* may be any one of the following messages: @ADD FILE NOT ASSIGNED OR CATALOGED @ADD ELEMENT NOT FOUND IN FILE @ADD FILE NOT MASS STORAGE @ADD OF EMPTY FILE FILENAME NOT ASSIGNED OR CATALOGED SYNTAX ERROR ON CSF$ IMAGE EMODE TYPE *octal* CODE *octal* FAC STATUS= *octal* Snaps: none
0102	*block-id filename* CKRS$ ERROR = *decimal* CKRS$ FUNCTION = *decimal* Snaps: CKRS$ packet
0103	*block-id* CONTINGENCY TYPE = *octal* ERROR TYPE = *octal* ERROR CODE = *octal* ERROR ADDRESS = *octal* Snaps: User's common bank data area with registers
0104	*block-id filename* INFO$ STATUS = *octal* INFO$ FUNCTION = *decimal info message* *info message* may be any one of the following messages: FILE NAME = *internal filename* FORMAT = *octal* DENSITY = *octal* PROCESSOR CODE = *code* TAPE CODE = *code* EQUIPMENT INDEX = *octal* EQUIPMENT BITS = *octal* FILES EXTENDED = *decimal* BLOCKS EXTENDED = *decimal* Snaps: none
0105	*block-id filename* MSCON$ ERROR = *octal* [MSCON I/O ERROR = *octal*] The I/O status encountered by MSCON$ is printed only if the MSCON$ error is 024. Snaps: MSCON$ packet, DREAD$ buffer, FITEM$ packet

Table 11-4. Checkpoint/Restart Messages (continued)

Associated Error Code (Octal)	Message/Description
0106	*block-id filename* TLBL$ ERROR = *octal* ¦TLBL$ I/O ERROR = *octal*¦ The I/O status encountered by TLBL$ is printed only if the TLBL$ error is 011. Snaps: TLBL$ packet, FITEM$ packet
0107, 0111	*block-id filename* READ IOW$ ERROR = *octal* Snaps: I/O packet, FITEM$ packet
0110, 0112	*block-id filename* WRITE IOW$ ERROR = *octal* Snaps: I/O packet, FITEM$ packet
0113	*block-id* FULL RESTART COMMON BANK LEVEL *decimal* NOT COMPATIBLE WITH *block or buffer id* LEVEL *decimal* *block-id* FULL CHECKPOINT COMMON BANK LEVEL *decimal* NOT COMPATIBLE WITH ER CKRS$ FUNCTION *function-id* LEVEL *decimal* Snaps: Checkpoint data block (112-word block), restart only
0114	*block-id* BLOCK LENGTH *decimal* NOT ALLOWED If this block length were allowed, the next block identifier would be outside the checkpoint data block (112-word block). This may occur if the checkpoint has been altered after the checkpoint is complete, or the checkpoint and restart levels are not compatible. Snaps: checkpoint data block (112-word block)
0115	*block-id* BLOCK TOO LONG BY *decimal* WORDS The checkpoint block is longer than can be used by this level of the common bank. This may occur if the checkpoint has been altered, or the checkpoint and restart levels are not compatible. Snaps: checkpoint data block (112-word block)
0116	*block-id* BLOCK TOO SHORT BY *decimal* WORDS The checkpoint block is shorter than can be used by this level of the common bank. This may occur if the checkpoint has been altered, or the checkpoint and restart levels are not compatible. Snaps: checkpoint data block (112-word block)
0117	*block-id* UNKNOWN BLOCK ENCOUNTERED The block identifier is out of sequence or is not recognizable.

Table 11-4. Checkpoint/Restart Messages (continued)

Associated Error Code (Octal)	Message/Description
	Snaps: checkpoint data block (112-word block)
0120	*block-id* FULL CHECKPOINT NOT ALLOWED FOR PRIVILEGED RUN
	Snaps: none
0121	*block-id filename* LOSS OF POSITION *block-id filename* LOSS OF POSITION, TAPE REWOUND
	Snaps: none
WARNING	*block-id filename* CANNOT BE RELOADED WITHOUT B OR M OPTION
	The test of this file was saved at checkpoint time but cannot be restored because this file is neither cataloged with the B-option nor assigned with the M-option. This does not apply to temporary files or tape files.
	Snaps: none
WARNING	*block-id filename* WAS PREVIOUSLY CATALOGED
	This file was to be cataloged at checkpoint time, but it cannot be restored to the to-be-cataloged state because it was previously cataloged at restart time.
	Snaps: none
WARNING	*block-id* BLOCK IGNORED
	This block is recognized but not used. This may occur if an error was ignored because of the A-option.
	Snaps: none
	run-id ACCOUNT NUMBER NOT FOUND
017	*run-id* CONTROL MODE CKPT NOT ALLOWED IN DEMAND
025	*run-id* IS USING TEST AND SET QUEUING, CHECKPOINT NOT ALLOWED
	run-id IS DEMAND, CK KEYIN NOT ALLOWED
023	*run-id* HAS OVERFLOWED PCT
022	*run-id* CANNOT BE CHECKPOINTED DUE TO INTERNAL ERROR
	Cannot get PCT buffer.
021	*run-id* IS USING WRITE ENABLED COMMON BANKS, CHECKPOINT NOT TAKEN

Table 11-4. Checkpoint/Restart Messages (continued)

Associated Error Code (Octal)	Message/Description
020	*run-id* CKPT COMMON BANK NOT CONFIGURED
	run-id NOT RESTART COMMAND PASSED TO RESTART FUNCTION
	run-id COMMAND FIELD NOT PRESENT IN INFOR
026	*run-id* HAS CHECKPOINT IN PROGRESS
024	*run-id* HAS ESI ACTIVITIES, CHECKPOINT NOT ALLOWED
016	*run-id* CHECKPOINT NOT CONFIGURED

12. Internal Executive Design

12.1. INTRODUCTION

This section is not required for intelligent use of the system. The intention is to provide additional insight into the key concepts and algorithms incorporated in the internal design of the Executive. The information presented is in no way intended as a complete description of internal logic in that many important aspects of the design are not mentioned. It is thought, however, that this section along with the internal design specifications either stated or implied in other sections (especially Sections 1 and 2, which discuss general capabilities and concepts) provides an overview of the total internal system design from the user programmer point of view. Design and capabilities relative to system generation and operator interface are not discussed, but they can be found in other Sperry Univac publications.

Sperry Univac reserves the right to make changes in the internal design without notice.

12.2. BASIC DESIGN PHILOSOPHY

To achieve the broad spectrum of functional and operational capabilities defined for the Executive in a reasonable amount of space and time, certain fundamental design criteria have been adopted:

- Wherever feasible and appropriate, individual components operate reentrantly. Where true reentrancy is not feasible, the individual component, rather than its requestor, is generally responsible for serializing its operation.

- Components and data types which are permanently resident in main storage are limited to ones for which nonresidency is either impossible or would impose unacceptable overhead.

- Central processor units (CPU) are, in general, treated equally. Components must be prepared to execute on any CPU. Exceptions are limited to certain maintenance functions, and to input/output (I/O) situations in which a particular CPU does not have a direct electrical path to a particular peripheral device. This approach allows maximum CPU utilization in multiprocessor systems.

- To assure timely response to real-time related events, interrupt lockouts and software interlocks on interrupt-related data are kept to a minimum. Wherever feasible and appropriate, real-time requests are given top service priority.

- To preserve generality, most Executive components operate as ordinary activities, called EXEC workers, and are managed in a fashion very similar to the management of user task activities. Exceptions are limited to functions for which switching is either impossible (for example, the dispatcher and interrupt queuing) or requires unacceptable overhead.

- To meet system integrity and program protection requirements, individual user service requests are validated to whatever extent is necessary to eliminate undesired interaction with the system or other users.

- Code required to support optional hardware components and software features is written such that it is not generated if the associated component or feature is not configured.

- To avoid wasting high cost resources like CPUs and main storage, data transfers involving low speed peripherals (for example, card readers, teletypewriters, etc.) are normally buffered. Generally, large-volume data is buffered to mass storage and low-volume data is buffered in main storage until sufficient data has accumulated to warrant loading a task to process it. This approach means that user tasks can be executed at rates commensurate with the overall power of the system, and need not spend long periods essentially idle in main storage waiting (usually many milliseconds) for short data transfers to complete. The symbiont complex constitutes the principal application of the philosophy.

12.3. EXECUTIVE MAIN STORAGE USAGE

12.3.1. General Layout and Discussion

12.3.1.1. 1106, 1108, 1100/10/20 Main Storage

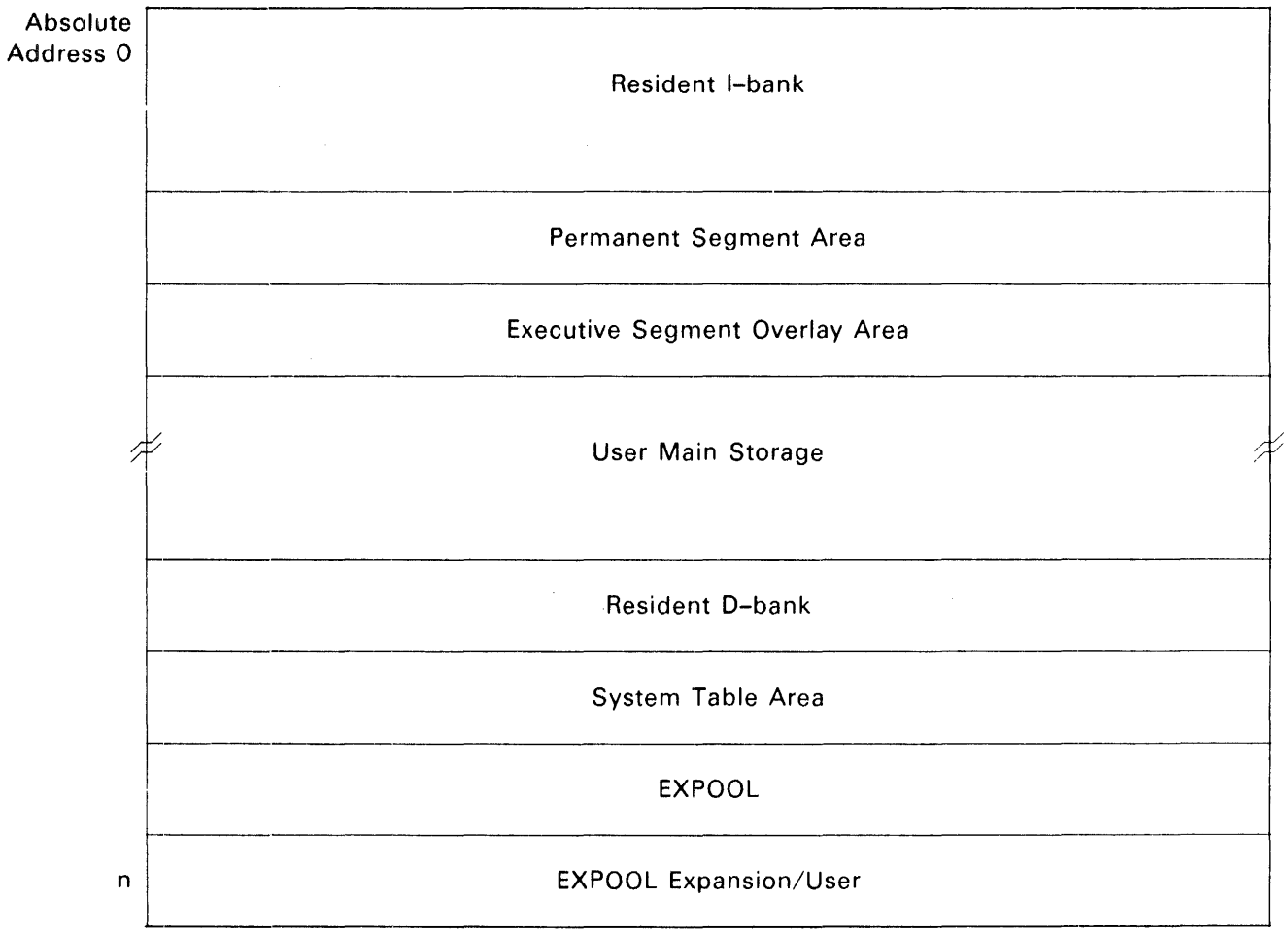


Figure 12-1. 1106, 1108, 1100/10/20 Main Storage

The individual areas of the general physical layout on 1106/1108, 1100/10/20 shown in Figure 12-1 are as follows:

1. The resident I-bank area includes the interrupt locations and permanently resident Executive instructions. In general, no data is stored in this area. Also, it does not include any routines which are referenced directly by Executive segments (nonresident routines) since the segments are also handled as I-banks.

2. The permanent segment area is devoted to holding certain Executive segments (transient or nonresident routines) permanently in main storage. Such segments, if any, may be selected at system generation on the basis of throughput benefit associated with a particular site's job mix and configuration.

3. The Executive segment overlay area is devoted exclusively to holding Executive segments (transient or nonresident routines). This area is at least as large as the largest segment. Design rules for Executive segments stipulate that a segment must not require any other segment to be loaded at the same time. Thus, at minimum, a serial operation is guaranteed.

4. The user main storage area is devoted primarily to user tasks and their associated program control tables (PCTs). Executive segments are also executed in this area when space permits; however, user tasks are always given priority over Executive segments. In a few cases, the Executive may acquire independent data buffers from the area, but these are normally acquired from EXPOOL (see number 6).

   When the system is heavily loaded, the high address end of the user main storage area may be used to temporarily expand EXPOOL space.

5. The resident D-bank area includes fixed (as opposed to pooled) data storage and those permanently resident routines which are referenced directly by Executive segments.

6. EXPOOL is the Executive data space pool. The EXPOOL mechanism is fundamental to Executive design, since the ability to rapidly and conveniently acquire data space is essential for efficient reentrant design. EXPOOL allocation is based on a powers-of-2 structure wherein buffer sizes are 3, 7, 15, 31, 63, 127, 255, or 511 words (one word is always used for buffer control). While this sizing may cause some waste in in-use buffers, the structure allows extremely rapid buffer collection, which is very difficult in pooling structures which provide a continuum of sizes. Briefly, whenever a buffer of a particular requested size is not available, a check is made for an available buffer of the next larger size; if there is one, it is split into two equal halves; if not, the process is recursively repeated for still larger buffer sizes until a buffer can be obtained. When a buffer is released back to EXPOOL, it is recombined with adjacent available buffers to make the largest possible buffer.

12.3.1.2. 1110, 1100/40 Main Storage

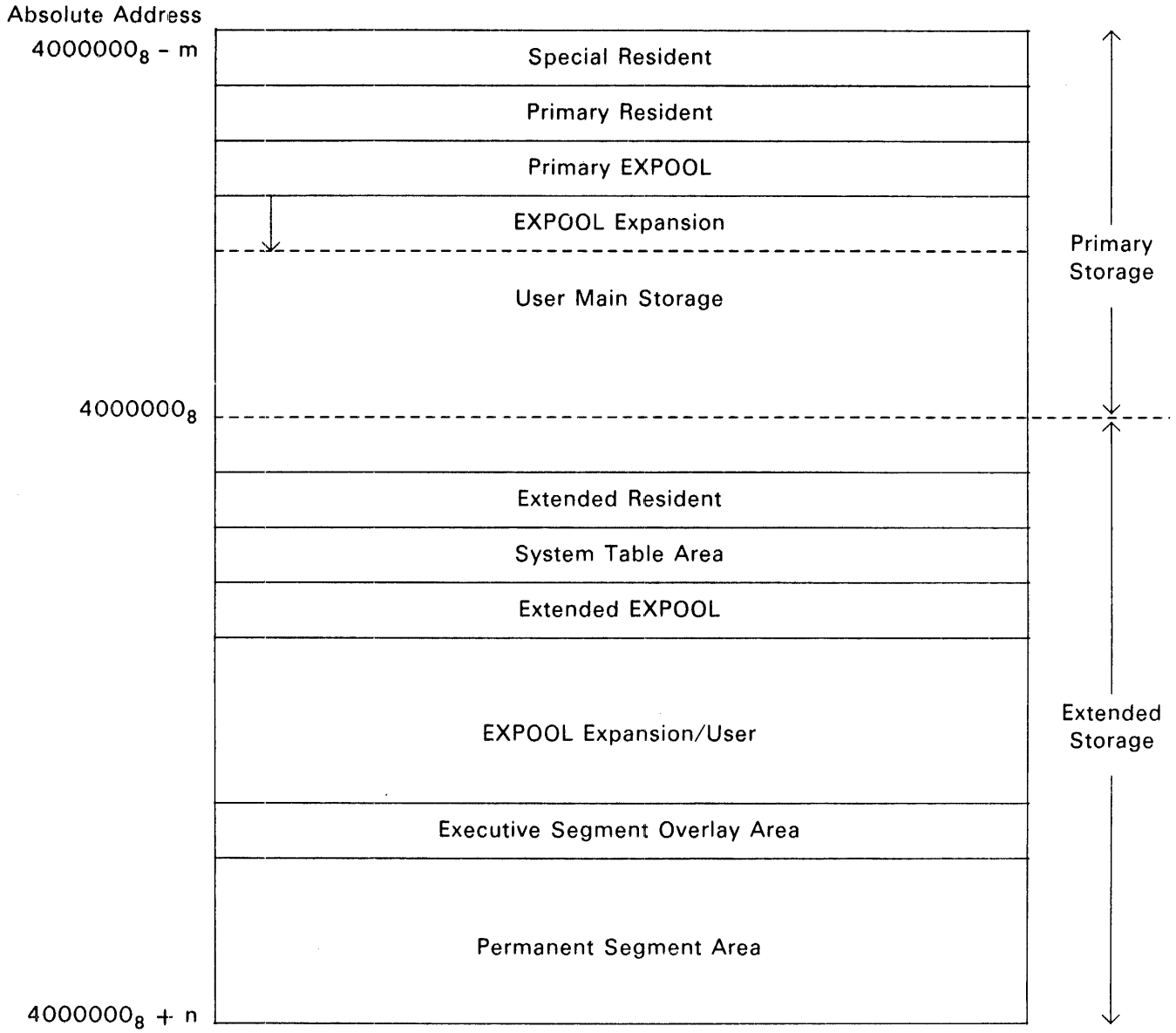


Figure 12-2. 1110, 1100/40 Main Storage

Differences from the general physical layout on the 1110, 1100/40 Systems shown in Figure 12-2 and the 1108 System layout shown in Figure 12-1 are as follows:

1. In all standard hardware configurations extended storage begins at address 4000000_8 and occupies consecutively higher addresses; primary storage ends at address 3777777_8 and occupies consecutively lower addresses. The layout, and the storage allocation algorithms attempt to place instructions and data that have a high reference frequency in primary storage and those of lower frequency in extended storage.

2. 4-bank relative addressing is used in the Executive, as follows:

   - Main I-bank covers Special Resident, Primary Resident and Primary EXPOOL
   - Main D-bank covers Extended Resident and Extended EXPOOL
   - Utility I-bank covers Executive segments. However, they use the same relative address space as the Special Resident, so the latter cannot be referenced directly by Executive segments.
   - Utility D-bank covers user PCTs.

3. The Special Resident includes the interrupt locations and high-use resident Executive code which need not be directly referenced by Executive segments.

4. The Primary Resident contains high-use resident Executive code which is directly referenced by Executive segments.

5. The Extended Resident includes low-use resident code.

6. User programs may be loaded into either primary storage or extended storage, depending on execution characteristics and user parameters. Normally, PCTs and Executive segments are allocated to extended storage rather than primary storage.

7. An alternative layout is possible, wherein the Extended Resident, Extended EXPOOL and Extended EXPOOL Expansion areas are moved down adjacent to the Executive Segment Overlay area. This arrangement allows allocation of large programs across the primary/extended storage boundary.

12.3.1.3. 1100/80 Main Storage

Figure 12-3 illustrates Executive storage placement:

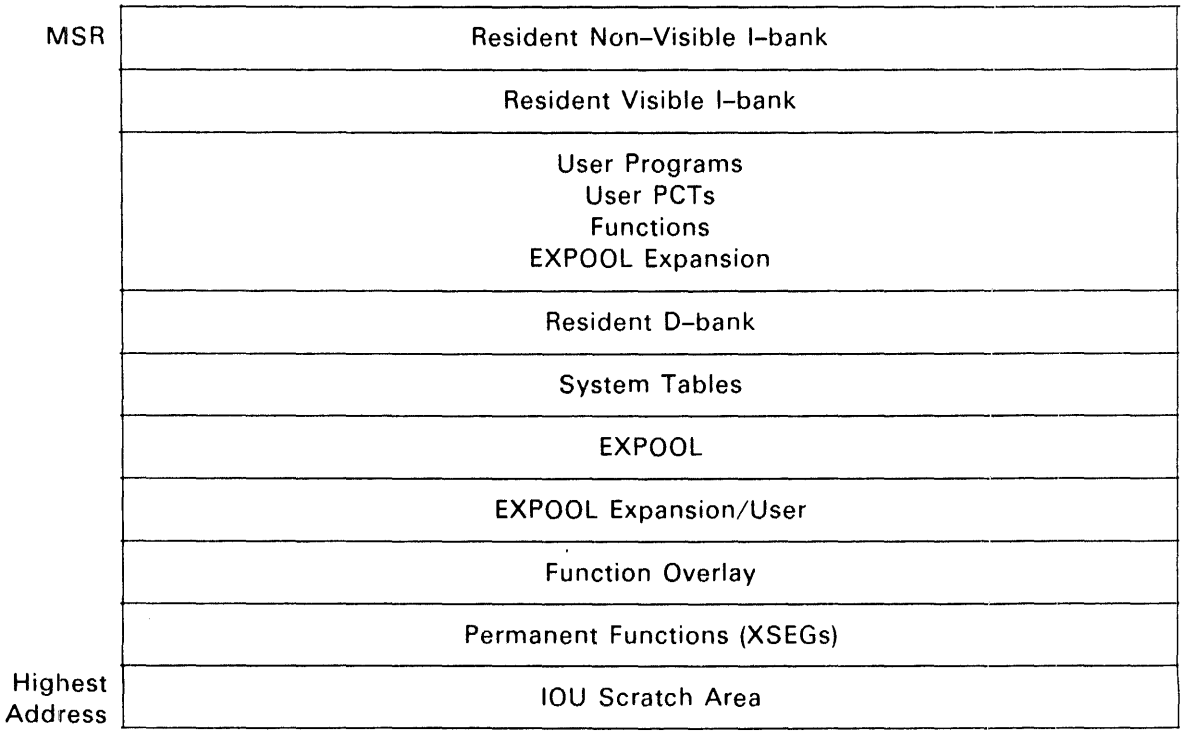


Figure 12-3. 1100/80 Main Storage

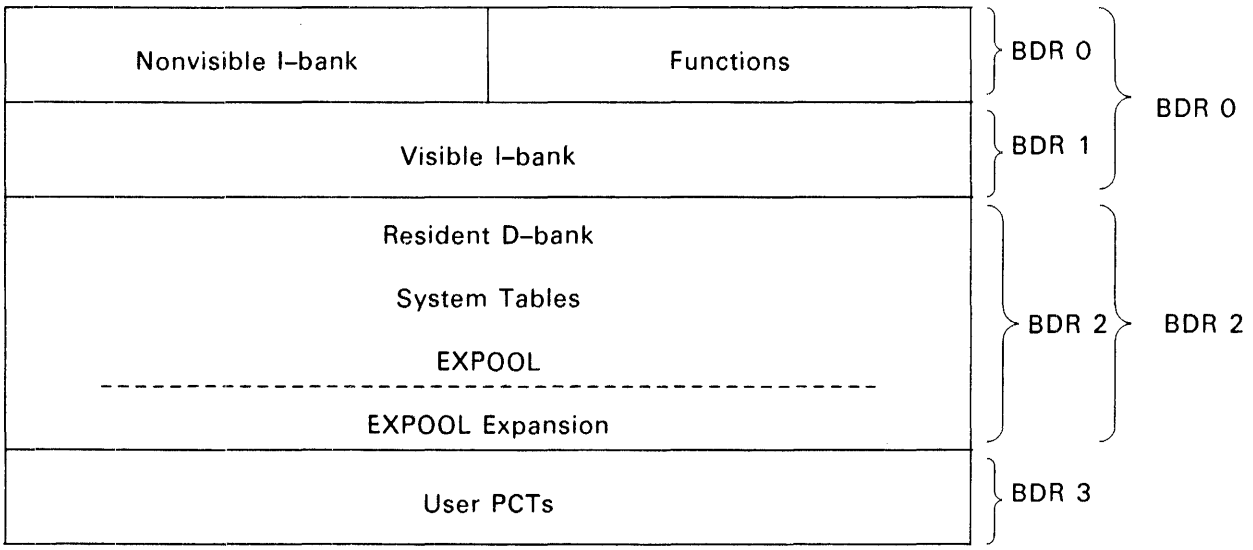
For 1100/80 Systems, bootstrap allows the EXEC to be put into SIU lower (absolute addresses 40 million octal or lower), SIU upper (absolute addresses 40 million octal or greater), or split between upper and lower.

Since there is no performance advantage to be realized by placing the Executive in different MSUs, there are no layered storage considerations on the 1100/80. This scheme allows user program space to be contiguous and facilitates EXPOOL expansion.

12.3.1.4. Executive Addressing Windows

The Executive may address any portion of main storage. Commonly required references are accomplished via relative addresses. Other references are accomplished via absolute addressing.

On the 1100/80, the BDRs which facilitate relative addressing are dedicated to provide the addressing windows required by different types of Executive coding. The following illustrates relative addressing windows and BDR dedication for the Executive:



This structure accommodates EXEC relative addressing needs as follows:

- EXMOD/ESIMOD Window

  BDR 0 – Non-Visible I-bank
  BDR 1 – Visible I-bank
  BDR 2 – D-bank, Tables and EXPOOL
  BDR 3 – User PCT

- Function's Window

  BDR 0 – Function I-bank
  BDR 1 – Visible I-bank
  BDR 2 – D-bank, Tables, and EXPOOL
  BDR 3 – User PCT

- Interrupt Preprocessing Window

  BDR 0 - Non-visible I-bank, Visible I-bank
  BDR 2 - D-bank, System Tables, EXPOOL
  PCTs are referenced via absolute addressing.

As well as supporting EXEC relative addressing needs, this structure also allows write protection of the I-bank and functions for most EXEC execution.

12.3.2. PCT Usage

Most data controlling the execution of a particular user run or task is maintained in the PCT. Such data is maintained in EXPOOL only if the data may be required while the task (including its PCT) is swapped out (for example, swapping control information) or if it may apply to another run (for example, information pertaining to cataloged files). This allows many runs (especially demand runs) to be controlled concurrently without an exorbitant load on EXPOOL. A prime example of this approach is activity control; the basic information concerning the status of an activity is kept in EXPOOL, but the activity environment, including control registers, is saved in the PCT.

PCTs have two logical parts: a fixed area containing data always needed to control a run or its current task, and a variable area. The space in the variable portion is pooled by a mechanism analogous to EXPOOL. Also, the PCT may dynamically expand from a normal minimum of four main storage blocks (01000 words per block) for batch runs and three main storage blocks for demand runs, up to the configured system maximum of 42 main storage blocks.

12.3.3. Definition and Residency of Components

Table 12-1 lists the Executive components that permanently reside in main storage. Table 12-2 lists the Executive components that are normally transient in main storage.

Table 12-1. Resident Components of the Executive System in Main Storage

Executive Component
Dispatcher and Activity Control
ER Interrupt Handler
Fault Interrupt Handlers
Test and Set Interrupt Handler
Executive Segment Controller
Clock Interrupt Handlers and Clock Management
Basic Internal Service Routines (EXPOOL, Access Word Check, and so forth)
Basic Dynamic Allocator
Basic I/O Control and Interrupt Queuing
Basic Communications Control
ISI Device Handlers
Console Handler
Symbiont/User Interface Routines (READ$, PRINT$, etc.)
Basic Symbiont Interface and Control
Basic Symbiont Device Control
Logging Control
TIP

Table 12-2. Nonresident (Transient) Components of the Executive System

Executive Component
Coarse Scheduler (Including Run Initiation)
Control Statement Interpreter
Run Termination
Control Statement Function (CSF$)
@START Control Statement Handler
Logging & Accounting
Checkpoint/Restart
User Task and Segment Loader
Dynamic Allocator Periodic Algorithm Adjustment
Program Expansion/Contraction (MCORE$/LCORE$)
Activity Error Termination
Task Termination
PCT Expansion
Activity Naming (NAME$)
Unsolicited Keyin Distributor and Handlers
Date Control and Keyin Handler
Console Patch Routine
Symbiont Probe
Symbiont Device Initialization/Termination
Basic Symbiont Device Control
@ADD Statement Control
Symbiont Forms Control
Symbiont File Manipulation (Alternates, Breakpoint, SYM, and so forth)
Symbiont Output File Queuing and Termination
Symbiont Operations Control (Repositioning, Error Handling, and so forth)
Facilities Inventory & Assignment
Facilities – I/O Interface
Mass Storage Space Allocation
Master Directory Control
Cataloged File Rollout/Rollback
Cataloged File Recovery
Tape Reel Control (TSWAP$ and TINTL$)
Tape/Disk Labeling
Communications Initialization and Termination
System Initialization
Program File Package

12.4. MULTIPROCESSING

The scheduling and CPU switching techniques used in the Executive system provide multiprocessing capabilities. The extension to multiprocessing leads naturally from the multiprogramming aspects of the system where many independent activities in user programs are available for execution at any instant. Within the Executive itself a similar situation exists where at any moment more than one independent EXEC worker requires CPU time.

Consider an Executive activity for a program that has an outstanding request for an I/O operation. When the I/O completion interrupt occurs, the Executive interrupts the activity at a point unknown to it. The activity environment is saved, the interrupt processed, and control returned to the activity.

The activity is never aware of the event, and would not be the wiser had the interrupt taken the execution to another CPU where the task was performed to process the interrupt.

In all multiprocessor configurations each processor interfaces to all storage; only one copy of the Executive is used.

The many runs being input to the system provide a number of tasks to be multiprocessed. Within any run, the individual tasks are executed in a serial manner as directed by the user. Among the many runs, the Executive typically uses the processors of the system to work on tasks of more than one run.

The Executive provides the ability, via forking to generate multiple activities, for a user to split a program into an arbitrary number of independent execution paths. Each activity is available for processing on any of the CPUs of the system.

The areas of Executive coding which reference common data and other areas with specialized coding methods must be protected from simultaneous execution, but many areas are open and multiprocessed as necessary.

12.5. SCHEDULING

12.5.1. General

The general scheduling technique used by the Executive removes any difficulty in the advancement of an installation into the use of multiprogramming.

The scheduling components of the supervisor are responsible for the control of facilities as well as the actual scheduling of runs and tasks. This includes both the assignment and release of facilities. There are five components within the system for handling the scheduling of runs and the tasks within runs; these are:

- Facilities Inventory
- Control Statement Interpreter (CSI)
- Coarse Scheduler
- Dynamic Allocator
- Dispatcher

Each routine is discussed individually in the following paragraphs.

12.5.2. Facilities Inventory and Selection

The facilities at the disposal of the Executive system include the I/O channels and all peripheral equipment attached to these channels, including available communications line terminals. Available facilities and their disposition are indicated at system generation; thereafter, the Executive assigns these facilities as needed and as available, to fulfill the facility requirements of all runs entering the system. The Executive maintains and continually updates inventory tables that reflect what facilities are available for assignment, and which runs are using the currently "in use" facilities.

Facilities inventory is a basic constituent of the scheduling section of the supervisor. The routines controlling the facilities while requiring a minimum of user-generated statements concerning operational requirements.

Devices such as magnetic tapes are normally assigned before run execution time because they cannot be shared by two or more runs and they normally require operator set-ups. Such devices are always released automatically by the termination of the run. However, they may also be released during the course of the run by the user.

Mass storage space is dynamically assignable and releasable by the user. The user is encouraged to do so whenever possible, since no relocations of information is necessary in allocating mass storage facilities (as may be the case for main storage).

Word-addressable devices (FH-432, FH-1782, and so forth) are allocated internally in sector-formatted granules (track, position). This allows the same space to be assigned as either word-addressable format or as sector-formatted mass storage, depending on user requirements at the particular time. A portion of drum or disk is set aside at system generation for the residence of the system and the processors. Normally, all user files which utilize mass storage during the course of a run are purged at the completion of the run. The space used for such files is returned to the pool of available facilities. If so specified, however, a file may be retained for future reference (cataloged).

Mass storage space is allocated in granules, with one granule equal to a track or a position (64 tracks). Given the equipment type from the @ASG control statement, the unit and area for the allocation of a granule of a particular file are selected according to the following hierarchy (only a simplified view of the algorithm is presented):

1. Allocate in a particular place on the unit which permits a contiguous allocation to a previous allocation for this file.

2. Allocate anywhere on the same unit as per the previous allocation.

3. If the file is cataloged, allocate on the unit containing the master file directory information for this file.

4. Allocate on the unit (of the desired equipment type) with the most availability.

5. Change the equipment type to next most desirable type and repeat step 4.

By use of the @ASG control statement, a user specifies the number of granules to be initially reserved for the file. Each dynamic request for additional space then results in the assignment of an additional granule. When using the system file control routines, the user does not have to request additional mass storage space because this procedure is taken care of automatically by the system.

When the coarse scheduler (see 12.5.4) selects a run to be opened, an initial facility synopsis is performed to determine if the run can be opened. If a requested facility is not available, information pertinent to the run and the reason for the hold are gathered in a queue. The queue has five sections based on whether the hold is for initial facilities, facilities between executes or facility requests from a demand terminal.

The hierarchy of the sections within the queue is as follows:

1. Executive requests
2. ER CSF$ requests
3. Facility requests from a demand terminal
4. Facility requests between executes from batch runs
5. Initial facility requests from batch runs

A held run remains in the queue until the hold mechanism is stimulated by the release of a facility. At this point, a search of the queue is performed based on the aforementioned hierarchy to determine if there is a held run which is waiting for the facility just released. If such a run is found, it is removed from the queue and a facility synopsis is again performed.

The operator is informed via a console display when the Executive has tried to open a held run the number of times specified at system generation. The operator may then remove the held run via keyin. From a demand terminal, the @@X T statement removes the demand run from the held status.

12.5.3. Control Statement Interpreter (CSI)

CSI is used by the Executive for interpreting the input runstream, and it is exercised by both the input symbionts and the coarse scheduler. It is also used by the CSF function to interpret control statement images passed to that function internally via CSF$ requests. For each control statement image passed for interpretation, CSI:

1. Format checks the statement.
2. Requests continuation statements (continuation character is treated as an illegal character if request was to CSF$)
3. Changes the external format of the control statement to an internally formatted table acceptable to the system

CSI is the single component of the supervisor that dictates the syntax of the Executive control language. CSI could be modified to accept this input different from that specified by the Executive control language as long as the interpretation presented functions known to the coarse scheduler and in the proper order and grouping. The coarse scheduler is the level at which the capabilities and functions of the system are defined. The coarse scheduler scans the input stream via READ$ requests in search of the next logical tasks or parameter on which it must act. If the next statement is a control statement, READ$ activates CSI and the internal table it builds is passed back to the coarse scheduler. The interface between CSI and the system is fixed, but input to CSI is fixed only in the sense that the Executive control language is defined to have a particular syntax.

12.5.4. Coarse Scheduler

The coarse scheduler has two basic functions:

1. Introduces new runs into a backlog queue and the updating of runs already in the backlog queue.

2. Processes control statements in an open run and the subsequent initiation of an Executive operation or user task which is specified by a control statement.

The first function is initiated when a new run is introduced to the system. The run may be introduced via an @START control statement, an @RSTRT control statement, or it may come through a symbiont input device. In all cases, the coarse scheduler performs legality checks on the @RUN control statement, and if the run is a batch run, an entry representing the run is built and inserted into the backlog queue which is more commonly known as schedule queue (SCHQ). Demand runs are not inserted into SCHQ after the legality check on the @RUN control statement. Instead, they are immediately opened.

The coarse scheduler also updates the scheduling parameters of runs in SCHQ. The update must be performed when one of the following conditions exists:

1. An S option hold, a batch symbiont hold, or a console imposed hold is to be removed from a backlogged run.

2. Scheduling parameters for a particular run or global parameters, are dynamically changed via a console keyin.

3. A backlogged run's priority is changed because a deadline time is imminent or the hold on a run which is imposed by a start-time must be removed because the start-time is imminent.

The first function also includes the selection of batch runs to be opened from SCHQ. The following rules and constraints govern this selection process:

1. No runs except special system initiated runs are opened if:

   a. the maximum number of batch runs (defined at system generation or defined by the operator) is already opened, or

   b. system overload is approaching (for example, EXPOOL or mass storage is near saturation).

   If the above test is passed and a global scheduling hold is set, only "removed" runs and "allowed" runs are considered for opening. An allowed run has priority over a removed run, and the allowed run with the highest priority is chosen first.

2. If a global hold is not set and test 1 is passed, a run is selected for opening on the basis of its current priority. Thus, a run is selected for opening only if its priority is the highest priority in SCHQ. If two or more runs contain the highest priority in SCHQ, the run which has been a candidate for opening for the longest period of time is selected. A run becomes a candidate for opening at the instant that it is free of all of the following hold conditions:

   a. a hold imposed by a start-time,

   b. an S option hold, and

   c. a batch symbiont hold.

If a run contains a deadline time as one of its scheduling parameters and the time when that run must be opened is approaching, that run's priority is adjusted so as to put it above the priorities of nondeadline runs.

After a run has been selected for opening, the second function, the processing of control statements, is started. The coarse scheduler reads the first set of facility control statements, links them together and sends them to facility inventory for processing. The reading and linking of facility-type control statements by the coarse scheduler is called a facility synopsis. All control statements except the conditional control statements and the @MSG, @LOG, @HDG, and @EOF control statements terminate the facility synopsis mode. These control statements are considered transparent during the facility synopsis mode.

The processing of the initial facilities block by facilities inventory can have one of three results:

1. rejected due to insufficient facilities, in which case the run is not opened, but rather placed in a facilities hold (see 12.5.2);

2. rejected due to an error in the control statement, in which case the run is fully opened and all statements up to and including the statement causing rejection are processed and printed along with any warning or error messages (includes transparent statements). The run is then terminated in error;

3. accepted.

If the facilities block is accepted, the run is fully opened and all statements in the block are processed as in (2), but run processing continues with the control statement that terminated facility synopsis mode. This statement is read and the operation it specifies is initiated for processing or the operation may be a user task in which a main storage request packet is built and sent to the dynamic allocator, thus directing it to load a program. When the EXEC worker has finished, or the user program has terminated, the coarse scheduler is again called to perform the following:

1. If a user program has just terminated, the abort and error indicators for the run are tested to determine if:

   a. the run should be aborted;

   b. the run should be terminated in error, in which case only @PMD control statements are honored; or

   c. the processing of control statements can continue in normal fashion.

2. The next control statement is read and the process is repeated until an @FIN control statement is read. Facilities inventory is next called to begin the run termination procedure.

NOTE:

Facilities control statements embedded in the runstream are blocked and handled in facilities synopsis mode in a fashion similar to that for the initial facilities block. The differences are that the run is already open and any hold is a between-tasks facility hold rather than an initial facilities hold.

12.5.5. Dynamic Allocator

12.5.5.1. General Overview

The function of the dynamic allocator (DA) is the dynamic allocation of main storage space to the current mix of tasks (programs), where "current mix" is determined by the particular group of runs presently being processed. Assuming that more than the maximum number of requests exist for the resources (main storage space and CPU time) of the central computer, it is the job of the dynamic allocator to make an equitable allocation of these resources in order to best serve the varied interests of all users. The allocation is based on the type of tasks (real-time, demand, and batch), as well as on the priorities and response times within a particular type. The basic principle under which the dynamic allocator operates is that the primary concern of any computing installation is the completion of batch runs at the required deadline (within the limitations of the operating environment), while at the same time attempting to maintain the required response times for demand users. For 1110, 1100/40 Systems, the DA must also keep primary and extended storage loaded as efficiently as possible. Within this dynamic operating environment, the dividing line between demand and batch programs is subject to constant change as emphasis is placed upon allocating time to batch runs approaching the required completion time.

In order to achieve the desired goals in system throughput and provide complete time-sharing capabilities, each of the program types has characteristics which differentiate it from the others for main storage allocation purposes:

- Batch Tasks – A batch task is the execution of a system processor or an object program in a nonconversational mode. Batch programs are ordinarily not removed from main storage to provide space for other batch programs while progressing toward completion. In some instances, a batch program is swapped because it has reached an impasse such as a self-imposed wait condition or a system or operator imposed facility wait condition.

- Deadline Tasks – A deadline task is a program which has a required completion time. Deadline tasks cause the swap of nondeadline tasks if necessary to achieve the run completion time requirement.

- Demand Tasks – Demand tasks are contained within a run entered from a demand remote terminal and generally thought of as being conversational in nature. Demand programs preempt batch as well as other demand programs for main storage allocation (within user-defined constraints). Frequent but relatively short periods of main storage residency are generally required to provide the desired terminal response time.

- Real-Time Tasks – A real-time program is a batch or demand program which is given preferential treatment because it is responding to real-time hardware. These programs are never subject to swap or relocation by the dynamic allocator and as such are positioned in main storage so as to have minimum impact on system throughput. The presence of a real-time program in main storage effectively reduces the amount of main storage that can be allocated between the other program types.

- Transaction Tasks – Transaction task allows a remote terminal to initiate the executions of a preregistered program at the central computer site. Once in execution, the transaction program has access to the full capabilities of the operating system.

12.5.5.2. Dynamic Main Storage Allocation

The dynamic allocator is a service routine which reacts in response to request from programs and other Executive routines. The DA consists of three independent functions. These are:

1. The DA proper, which selects the highest priority request from a main storage queue and determines the most economical method of satisfying the request.

2. Core contents control which carries out the requests in the manner decided upon by the DA. This is the function which initiates the I/O requests for program swapping and loading.

3. DA periodic adjustment (DAPA), which reviews main storage usage periodically and makes appropriate adjustments in priorities and system algorithms.

Each entry on the main storage queue is a service request for a program and has associated with it a request type and a priority. The priority is a function of program type, @RUN control statement priority, program characteristics, and request type. Entries are placed on the main storage queue in response to program requests and Executive requests. The request types are:

1. Swap Out – This is requested in order to position real-time programs, to expand the program, or to service a higher priority program. A program is swapped for a higher priority program only if it is in a long wait state (for example, wait for tape mount, TWAIT$ of more than one-half second, or wait for terminal response) or if it has used up its first core quantum and the other program is not batch.

2. Initial Load – This is the request to load a task (absolute element) and is the result of an @XQT or processor control statement.

3. Reload – When an active program is swapped due to higher priority requirements, the request to reload that swapped out program is made at the time of the swap out. For a waiting program, the reload request is made when the wait condition is removed.

4. PCT Expansion – The PCT must be expanded as the result of some control statement or program request.

5. Bank Expansion – Service a MCORE$ request.

6. Bank Load – The program is requesting a dynamic bank or a common bank which is not already in main storage.

7. Real-time Relocation – Reposition a real-time program toward a main-storage boundary.

8. Bank Move – This is a request to move a bank from external primary storage (1110, 1100/40) to better utilized primary storage.

Whenever a bank is to be swapped, it is allocated area within a system file, SWAP$FILE. Write-protected banks are swapped once only as it is assumed that they do not store into themselves. When a main storage area used by a write-protected bank is required, the space is used and the next time the write-protected bank is requested, it is loaded from swapfile. The swapfile space for non-write-protected banks is released after the bank is reloaded. When a program is reloaded after being swapped, the PSRs (processor state registers) and SLRs (storage limit registers) of all activities are recomputed.

When main storage is allocated for a program, the DA attempts to position I-banks and D-banks in different main storage modules in order to avoid main storage reference conflicts. Each program is also loaded as closely as possible to the extremes of available main storage in order to reduce fragmentation of the available space. Real-time programs are always loaded at the absolute extremes of main storage, except for the Executive and other real-time programs, even if this involves swapping other programs.

For 1110, 1100/40 Systems, the DA tends to keep primary storage full and actively executing, with extended storage being loaded whenever possible to keep all processors active. Compute intensive programs will be placed in primary storage, perhaps after being staged through extended storage. Programs which do less compute tend to be executed in extended storage.

Executive transient (nonresident) routines are also loaded into the same main storage area as programs when space is available. However, programs are never swapped to satisfy an Executive routine main storage requirement. The DA also controls the loading of these transient routines.

Whenever a system transient routine completes its current operation, the main storage area it occupies is not actually released, but is placed in a release-if-necessary condition. Such a routine is therefore still available for use, if necessary, until the main storage space it occupies is required for some other operation. If the transient routine is requested again before such an event, its main storage is returned to the in-use condition. Each such transient routine has associated with it a "sticking priority", so that the more frequently a transient is used, the longer it tends to retain its main storage space after each period of operation. This procedure prevents unnecessary loading of transient routines, since they remain in main storage as long as it is possible to do so without interfering with the over-all operation of the system.

Definitions:

Core Queue – This is the priority structured list or queue into which program requests for main storage allocation are placed. It is this list from which the DA selects the next task to receive the desired allocation.

SUPs – Standard Units of Processing. SUPs are computed by the following formula:

$$SUPs = CPU + \Sigma_i (\text{wds-transferred} * \text{rate} + \text{nbr-accesses} * \text{average access time}) + ER$$

where:

CPU = Total CPU time

i = I/O Device type being considered

ER = Control Statement and ER charges

Core Block SUPs – These are SUPs multiplied by program size in core blocks and are accumulated in the PCT for accounting purposes. They are also the basis for demand/batch sharing.

Core Quantum – The core quantum is the number of SUPs which a program is allowed to accumulate before it becomes available for swapping. The core quantum is computed each time the program is loaded. A program is not swappable until the first quantum is exceeded or it enters a voluntary wait state.

Core Priority – This value defines a program's position on the core queue and dictates how other programs may cause it to be swapped. For demand programs, the core priority is reduced one level and the quantum doubled each time a core quantum is exceeded. The priority is reset to the highest allowed for the program when the program responds to the terminal (via READ$ or TREAD$ requests).

12.5.5.3. Demand/Batch Sharing

The following parameters provide basic input to the demand/ batch sharing algorithm. They are alterable by the operator:

- DMIN – Minimum percent of computer usage guaranteed to demand, if requested.
- DMAX – Maximum percent of computer usage allowed to demand runs when batch runs are requesting service.
- DINC – Increment in percent to be added to DMIN for each active demand run. The percentage used is DMIN + (number of active demand runs *DINC), except that if its value exceeds DMAX, DMAX is used.

Total core block SUPs are accumulated by program type. The number of core blocks swapped is also accumulated. The blocks swapped are multiplied by average swapping time to convert to core block SUPs and the result is added to the demand core block SUPs. These values are used to determine the necessity of any sharing adjustments.

The goal of demand/batch sharing is to provide for maintaining some minimum batch throughput. The values used to control the sharing, represent that percentage of the normal batch throughput capability which may be used for demand processing. The exact correspondence between the percentage and throughput varies somewhat with program mix but should still maintain an approximately linear scale.

If demand programs are exceeding the percentage allotted to them and batch programs are waiting, the highest priority deadline or batch programs are given a larger than normal quantum and are queued ahead of all demand programs. Demand programs do not cause the swapping of deadline programs unless the allotted demand minimum percentage is not being met.

12.5.5.4. Time Sharing

Time Sharing is an attempt to equally distribute facilities among demand users. For a program whose size in core blocks is given by s, the cost of loading is proportional to s^2. The I/O transfer time is proportional to s and s core blocks are tied up for the duration of the transfer.

The queuing algorithms are designed such that a program at level n is loaded twice as often as a program at level $n+1$. The highest level allowed for a demand program is then determined as a function of s^2.

This is the level at which the program initially receives control and the level to which it is raised when requesting input from the terminal.

The quantum allocated to the program is computed from the average time required to load the program. The quantum is constrained to be at least equal to this time. It is adjusted by @RUN priority and is doubled for each level below the highest level allowed for the program.

High priority programs must not be allowed to dominate the system to the exclusion of all others. In order to appropriately allocate system resources to all users, the following algorithm is applied to the core queue:

An effective time on queue is determined for the oldest request at each demand level:

$$T_E = (T_A - F1) * F2 ** L_D$$

where:

T_E – Effective time on queue

T_A – Time request has actually been queued

F_1 – Parameter (currently 1 sec.)

F_2 – Parameter (currently 1.9 = 90% increase)

L_D – number of levels from lowest priority

If $T_A < F1$, then $T_E = T_A$

The request with the longest effective time on queue is always selected for execution. This results in quick allocation for higher priority programs which have been on the queue for some time. It guarantees eventual allocation for low priority requests.

12.5.6. Dispatcher

The dispatcher has prime responsibility for controlling CPU usage. In a broad sense, dispatching includes all CPU control decisions; more commonly it applies just to switching among activities.

The CPU usage algorithm may be characterized as "pure preemptive". That is, if some process (typically an activity) having a certain priority is using a CPU and some event occurs requiring performance of a higher priority process, then the lower priority process is immediately preempted in favor of the higher. Conversely, if the new process is of lower or equal priority, it must wait until no higher priority demands for CPU service exist.

Two basic classes of CPU usage are defined: interlock and switchable.

12.5.6.1. Interlock Processing

Interlock processing is logically noninterruptable and nonswitchable. That is, once an interlock process begins, it is executed to completion and cannot, for example, be suspended and switched to another CPU or interleaved with other interlock processes of the same level. Interlock processes typically use the Executive control registers (a privileged duplicate set of registers not usable under guard mode.) They always have higher priority than any switchable process (activity). Three levels of interlock processing are defined:

1. Interrupt Processing – This is the highest level in the system, since the hardware locks out further interrupts on occurrence of an interrupt. Generally, interrupt preprocessing involves merely capturing essential information concerning the interrupt and recording it (typically in a queue) for subsequent processing (see 12.7).

2. ESI Processing – This is the next highest level (see Sections 9 and 10).

3. Executive Interlock – This is the lowest level of interlock processing. Most interlock processing occurs at this level. It includes the main dispatcher, internally specified index (ISI) interrupt processing, clock control and non-ESI (externally specified index) internal interrupt handling (ERs, Test and Set, and program faults). The switching routine of the main dispatcher is exercised only when no other interlock level processing remains.

The hierarchy among interlock levels is achieved by reserving some of the Executive registers for each level (Executive interlock is given all but a few).

12.5.6.2. Switching

All processing below interlock levels is done by activities. The selection and control of multiple activities competing for CPU time is the principal concern of the dispatcher.

For switching purposes, activities are grouped into six types, and within each type there are up to 35 levels, numbered in order of priority from 1 to 35. Level 1 of each type is reserved for interrupt activities. The six activity types are:

1. High EXEC Workers – This type is relatively scarce and is generally limited to those EXEC workers which must run above real-time activities; for example, the abort routine.

2. Real-time User Activities – See Section 10.

3. Low EXEC Workers – Includes most Executive workers and all transient routines.

4. TIP Activities

5. Deadline Batch Activities

6. Demand and Batch Activities

Real-time and Executive activities are switched with "infinite quantum". That is, an activity may execute indefinitely without "supervision" at the same level for as long as it wishes. Level control is the responsibility of the activity; no automatic level manipulation is performed by the Executive (as is done for demand and batch, described next). For real-time and Executive activities, switching priority is simply by type (high EXEC above real-time above low EXEC) and by level within type.

TIP, demand, and batch activities (type 4, 5, and 6) are switched in a much more complex manner. The algorithm to be described has the fundamental objective of maximizing throughput by the strategy of:

- giving high priority for short periods to activities that request synchronous I/O service, and
- giving lower priority for longer periods in proportion to the amount of computation an activity does.

This strategy is intended to maximize utilization of peripheral equipment while reducing switching overhead for computational processes.

The demand/batch CPU switching algorithm operates as follows:

- Priority is determined by the Switching Level Configuration. The standard configuration selects TIP followed by deadline and then demand/batch.
- Associated with each level is a quantum of CPU time. Standard quantum values for various levels are determined by the speed of the machine. These values are set subject to instrumentation. Quantum size increases as the level increases.
- Initial program activities start at level 2.
- As an activity executes, the amount of CPU time it consumes is accumulated in a quantum-used cell. When quantum-used exceeds the assigned quantum for the activity's level, the activity is dropped one level in priority, its quantum-used is cleared, and it is assigned a new quantum (twice as long). If the bottom level is reached, the same length quantum is assigned at the same level. In either case, the activity is placed at the end of the list of activities to be switched for the new level (and type). If a program has an initial core quantum, the bottom level is 4; otherwise, bottom level is 7.
- If an activity issues a synchronous request (ER) for service that involves I/O, the activity is raised. If the program has an initial core quantum, the activity is raised to level 2 and assigned a full level 2 quantum. If the program does not have an initial core quantum, the activity is raised to level 4 and assigned a full level 4 quantum.
- When an interrupt activity (level 1) is activated upon completion of its associated I/O request, it is issued a level 1 quantum. Expiration of that quantum, or any ER, causes the interrupt activity to be dropped to the level of the original activity.
- Forked activities receive the same quantum and level as the original activity.
- Quanta are timed with the real-time clock. Whenever an activity is given control, the time remaining for its quantum is computed and the real-time clock is set to interrupt no later than the computed time of quantum expiration.

Once any level control operations are performed, switching consists of simply taking the highest priority activity from a list, called the switch list, of all activities currently requiring CPU service. If the selected activity is of higher priority than the currently executing activity (if any), the latter is suspended and control switches to the selected activity. Otherwise, no switch occurs. If there are no activities requiring CPU time, the dispatcher goes into an idle loop until an interrupt occurs or, on multiprocessor system, an activity appears on the switch list. Note that the dispatcher is reentrant and can operate simultaneously on multiple CPUs by protecting the common switch list data at appropriate critical points.

12.6. CLOCKING

The clocking routines make provisions for clock usage by an object program as well as by the system. These routines are available to the user by means of ERs. The clocking routines serve as the basis for all accounting and logging functions, as well as a source of control for many real-time applications.

12.6.1. Real-Time Clock

The real-time clock routine is used by the system for timing various activities such as I/O functions, operator responses, and CPU usage (non-1100/80) time for each run. This routine is also used by the system to force interrupts after variable amounts of time so that such events as nonresponsive and search I/O functions, unbalanced usage of CPU time, etc., are detected. The frequency of interrupt depends on the needs of the system. The routine is designed so that many events may be simultaneously timed, and many interrupts may be simultaneously requested.

12.6.2. Day Clock

The day clock routine is used by the system to maintain an accurate, standard time. This time is used by all processors for annotating listings, by the file control system for maintaining historical information about all files, by the accounting and logging routines for time-tagging events, detection of nonresponsive I/O devices, as well as by other routines for various functions. The 1106/1108 day clock updates every 600 milliseconds and provides an interrupt every 6 seconds. The 1100/10/20/40/80 and 1110 update every 200 microseconds and provide an interrupt every 6.5536 seconds.

12.6.3. Quantum Timer

The quantum timer is used to time CPU time usage for runs on 1100/80 systems.

12.7. INTERRUPT HANDLING

The interrupt handler either queues an interrupt or routes control to the appropriate routine to handle the interrupt. Interrupts are received from either the control section of the CPU or from a peripheral subsystem. The interrupts fall into five general categories as follows:

1. Input/Output Interrupts
2. Clock Interrupts
3. Interprocessor Interrupts
4. Hardware Fault Interrupts
5. Program-Generated Interrupts

The handling of the real-time and day clocks is discussed in 12.6.1 and 12.6.2. This paragraph discusses the handling of the other four classes of interrupts.

12.7.1. Input/Output Interrupts and Queuing

When an I/O interrupt occurs, the type of interrupt and the channel on which it occurred are either queued or routed to the appropriate processing routine. The three types of I/O interrupts are:

1. ISI and ESI external interrupts
2. ESI and ISI data termination (monitor) interrupts; on 1110, 1100/40/80 Systems, ESI tabling occurred and table full interrupts

3. ISI function termination (function monitor) interrupts

ISI interrupts are queued if:

- higher priority interrupts are already in the queue;
- another interrupt is being processed by an interlock level routine; or
- the CPU was interrupted while operating in the ESI mode.

ISI interrupts are removed from the queue by channel priority, channel zero having the highest priority.

ESI interrupts are queued if all CPUs are currently busy on other ESI-level work. ESI interrupt handling is further discussed in Section 9 and 10.

12.7.2. Interprocessor Interrupts

Interprocessor interrupts are issued within the Executive via the privileged Initiate Interprocessor Interrupt (III) instruction. This feature allows one CPU to divert another CPU for the following reasons:

1. Issue an I/O request to a device which cannot be accessed by the first CPU (due to lack of direct electrical path).
2. Deactivate an activity which is currently executing on the other CPU; e.g., for program suspension prior to swapout.
3. Process outstanding ESI interrupts in order to distribute the ESI load over the entire system, in the event that the initiating CPU is already busy processing an ESI interrupt.
4. Inform another CPU that a high priority task needs control. Used when the other CPU is idle and the current CPU is activating an activity, and when the other CPU is doing low priority work and the new activity is real time.
5. System start-up.

Prior to issuing the III instruction, the sending CPU sets a lock (Test and Set) and stores a function code in a common cell. When the receiving CPU is interrupted, control is routed to the interprocessor interrupt handler. This handler retrieves the function code, clears the lock, and routes control according to the function code, which defines some action to be taken under one of the five categories previously discussed (see 12.7).

12.7.3. Hardware Fault Interrupts

12.7.3.1. Storage and Control Register Parity Error Interrupts

The type of interrupts in this category are:

1. Storage Parity Errors
2. Input/Output Data Parity Errors and Channel and IOAU Interface Parity Errors

3. ICR (Control Register) Parity Errors

4. Access Control Word Parity Errors

Object programs may register a contingency routine for Storage Parity Errors or ICR Parity errors to be entered if such an error is encountered while the program is in control. If the object program has not registered a contingency or if the Executive is in control, an interrupt of this type causes the system to halt. The only way to recover from these errors is to reload the system.

12.7.3.2. Power Loss Interrupts (1106/1108 only)

When a CPU receives a power-loss interrupt, three steps are taken to prepare for the total loss of power:

1. Save the control register contents of the interrupted CPU.

2. Disconnect all I/O channels of the interrupted CPU (1106/1108 only).

3. Go into a Jump Greater and Decrement (JGD) loop and wait for the total loss of power.

If the CPU is still running at the end of the JGD loop, a transient power-loss interrupt has occurred; and the following steps are taken to recover the CPU:

1. Restore all control registers.

2. Reactivate all previously active ESI channels. In the case of ISI channels, an I/O TIMEOUT message occurs and an operator answer reissues the command on the channel.

3. If a user or Executive activity was interrupted, set up the interrupted activity's PSR and SLR and return control to the activity. If the above is not true, return control to the Executive at its interrupted point.

In the event of a real power loss, the operator can recover the CPU without rebooting the system.

When the CPU is started, according to the power loss recovery procedures, all internal registers are reset and the recovery procedure for a transient interrupt is followed.

12.7.4. Program-Generated Interrupts

This class of interrupts includes all those which occur immediately as the result of program instructions (in some cases, these may occur inside the Executive). There are three subclasses of program interrupts:

1. Program Fault Interrupts (see 4.9):

   - Illegal Operation
   - Guard Mode Fault
   - Undefined Sequence

2. Arithmetic Exception Interrupts (see 4.9):

   - Floating-Point Overflow
   - Floating-Point Underflow
   - Divide Fault

3. Programmed Interrupts:

   - Executive Request (see Section 4)
   - Test and Set Conflict (see 4.3.4, and Section 10)
   - Breakpoint (1110, 1100/40/80 only, see Section 4)

12.8. CATALOGED FILE RECOVERY

When the system is reloaded following a hardware or software failure, the Executive reconstructs the master file directory of all cataloged files. The cataloged file recovery routine (CATFR) is executed during the initial phases of system reloading following the determination of the current hardware status.

All master file directory information on all mass storage units is read and verified, the look-up table and link addresses are recalculated, and mass storage is reallocated. These operations are performed on a unit by unit basis in order to minimize the amount of information which is lost should a unit be lost to the configuration or be partially overwritten.

In order to perform these operations as efficiently as possible, file recovery utilizes all available main storage. Independent parallel I/O operations are performed on all available mass storage channels to read in the master file directory tracks. Directory allocation sectors (DAS) describe the allocation of the master file directory tracks. If a mass storage unit's DAS chain is in error, a master file directory degradation scheme allows the recovery to continue, recovering as much as possible. If an individual track address within a DAS is in error, the track address is removed and file information within the track is lost. If the nth DAS in the unit's DAS chain is completely in error, the chain is truncated at the n - 1th DAS and file information within directory tracks in the unit's nth or greater DAS is lost. If the first DAS is in error, the operator can either stop the boot or, upon responding correctly, cause file recovery to either down or initialize the unit, losing all files or parts of files residing on that unit. Whenever file recovery encounters a bad DAS or a bad directory track address, the operator is notified via a console message and given the option to continue the recovery or stop the system.

After main storage is filled with master file directory items, the file validation phase begins. Each file is processed independently and multiple validation activities are used to perform this operation as rapidly as possible. The master file directory information for each file consists of linked items. If a forward link address or DAD table entry points to a downed unit or is a nonexistent address, the address is removed, the granule or last part of the file is lost, and the file is marked disabled in a special way ("hardware disabled") such that an attempt to assign the file results in a FACILITY REJECT message unless the Q option is used on the @ASG control statement.

If the file was assigned write-enabled at the time of the system stop, the file is marked disabled (standard disable) as it was possibly in a state of flux at the time of the stop. An assign of a disabled file is accompanied by a FACILITY WARNING message. Files which were still in the to be cataloged state, marked to be dropped, or whose FORWARD ITEM links were bad are dropped.

Following the verification, the granule counts are updated for the number of granules recovered and the file is entered into the look-up table, which is used by the facilities mechanism of the Executive.

In addition to recovering as much of the files as possible, file recovery's validation ensures that all information required by the Executive is present and correct. Unless some error is detected, the operation of file recovery is transparent to the user.

See Section 7 for further information relating to disabled files.

12.9. REMOVABLE DISK RECOVERY/REGISTRATION

When a removable disk pack is mounted, the Executive ensures that the master file directory (MFD) accurately reflects the current state of all cataloged files on that pack.

All MFD and disk directory information for all removable disk files on the disk is verified and the allocation bit maps on the pack are rebuilt.

In order to perform these operations as efficiently as possible and yet not impact system response, swappable data space is allocated. This data space is used primarily for track and record size I/O buffers to minimize the need for sector I/O.

If errors are detected on the pack, the operator is given the option of taking a dump, continuing the registration as best possible, or aborting the registration and those users queued for the pack. In addition to recovering as much of each disk file as possible, the validation ensures that all information required by the Executive to reference that pack is correct.

To ensure that the pack is currently registered with the Executive, the user must assign a temporary file on the disk pack before trying to reference any cataloged file on the pack.

For example:

```
@ASG,T     TEMP,F40,PACK-ID
@XQT
@FREE      TEMP
```

5. Symbiont Interface Requests

5.1. INTRODUCTION

The Executive System contains a set of routines which provides an interface between the user and any supported unit record device. This set of routines is called the symbiont complex. These routines can be divided into two logical groups:

- Symbionts (also called device routines)
- Symbiont interface routines

5.1.1. Symbionts

Symbionts (device routines) are available for all standard equipment. Supported equipment includes such onsite devices as high speed card readers, punches, printers, UNIVAC 1004 and UNIVAC 9000 Series Systems; such batch remote site terminals as UTS 700 UNIVAC 9000 Series Systems, UNIVAC 1004, and DCT 2000; and such demand remote terminals as Teletype models 33 and 35, UNISCOPE 100/200, UTS 400 Display Terminals, DCT 475, DCT 500, DCT 524, and DCT 1000. For all batch devices, data is buffered in SDF using mass storage to provide an effective linkage between the high speed of the CPU and the low speed unit record devices. Due to the conversational nature of demand processing, input data from demand terminals is not buffered to mass storage except when specifically requested, as by @@INQ, @@FUL, or paper tape input.

In the system generation, one or more output devices are associated with each of the input devices. This logical linking of output to input devices is called "device association" throughout this section. The result of this association is that output files created as the result of an execution are normally output on only one of the devices associated with the input device which initiated the runstream. In cases where an output device is unavailable or busy, or where a specific output device is desired, the association can be overridden for a specific file by means of the @SYM control statement.

Input to the system is separated by the @RUN control statement. As each @RUN control statement is encountered, a run file is created and information is extracted by the coarse scheduler for run scheduling. The input symbionts also interpret the @ELT,D, @DATA, @END, @FILE, and @ENDF control statements to determine if an @RUN control statement is the beginning of another input run stream or part of a file or element in the current runstream. The @COL and @ENDCL control statements are interpreted to determine if the mode of the card reader should be changed.

All files created or processed by the symbiont complex are in SDF (see Volume 3-11.2.3) and can be directly processed by either the input interface routines or by the output symbionts.

Functions which control the format of the output are inserted into the symbiont output file by means of an Executive Request. These functions vary according to the output device to which the file is being directed. As the control parameters are submitted, they are placed into the appropriate output file and interpreted when the file is being processed by a symbiont (see 5.4).

The data in input files created from ASCII devices is in ASCII, and the data in input files created from Fieldata devices is in Fieldata. The user may request data from these files in either ASCII or Fieldata (see 5.3). Data which is in ASCII is converted to Fieldata for Fieldata devices and requires no translation for ASCII devices. The converse holds true for data which is in Fieldata. This manipulation of data requires no special action by the user other than to make the proper Executive Request as described in this section. (See Appendix D for translation tables.)

5.1.2. Symbiont/User Interface Routines

The symbiont user interface routines provide for data transfers in either Fieldata or ASCII. A complete set of Executive Requests is provided for each mode. The data transferred is always Fieldata when the Fieldata requests are used and is always ASCII when the ASCII requests are used. The user interface routines are available through the following Executive Requests:

Fieldata Executive Requests	ASCII Executive Requests
READ$	AREAD$
PRINT$	APRINT$
PUNCH$	APUNCH$
READA$	AREADA$
PRNTA$	APRNTA$
PNCHA$	APNCHA$
PRTCN$	APRTCN$
PCHCN$	APCHCN$
PRTCA$	APRTCA$
PCHCA$	APCHCA$
TREAD$	ATREAD$
CLIST$	CLIST$
SYMB$	SYMB$

When a letter A appears as the last alphabetic character, the request pertains to an alternate file (defined in succeeding paragraphs). The letter A appearing as the first character indicates an ASCII operation. ASCII and Fieldata Executive Requests may be interspersed in any order. For each Executive Request, the user specifies the storage area in the program for the data transfer. In addition, when using Executive Requests for alternate files, the user must specify the internal filename in Fieldata.

The system automatically initiates three symbiont files, allowing three normal operations as follows:

Run file (READ$ file) - contains input images accessed by means of READ$ or AREAD$
Print file (PRINT$ file) - contains output images produced by PRINT$ or APRINT$
Punch file (PUNCH$ file) - contains output images produced by PUNCH$ or APUNCH$

Each of the three basic interface functions (read, print, and punch) is capable of multiple file operation. The user may define files other than the three automatically initialized by the system. The user may assign a file and direct the normal print or punch output to this file by means of the @BRKPT control statement (see 3.6.2). The user may assign a file and direct only specific print or punch output to this file by means of the alternate output Executive Requests (such as the PRNTA$ request). The user may also assign a previously created file of input images and read these images in the normal mode

by prior use of the @ADD control statement, or directly input from the file by means of alternate input Executive Requests (such as the READA$ request). These user-defined and assigned files are called alternate files.

The @ADD control statement is used to direct the READ$ or AREAD$ requests to obtain images from the file or the element indicated by the @ADD control statement instead of images from the system initiated run file. Subsequent READ$ or AREAD$ requests obtain images from the @ADD file or element until it is exhausted, at which time images are again obtained from the system initiated run file. Nesting of @ADD control statements is permitted.

The @BRKPT control statement is used to direct PRINT$/APRINT$ or PUNCH$/APUNCH$ requests to place images in a file defined by the @BRKPT control statement instead of the system initiated print or punch files. Images continue to be placed in the user specified files until another @BRKPT control statement is encountered. During run termination, the normal print image stream is always returned to the system-initiated print file. The @BRKPT control statement also may be used to close user-defined alternate files.

The output control requests, such as PRTCN$, provide specific control information describing output formatting to the device routines. The output control requests also provide a means of advising the device operator of any special action required.

A more detailed discussion of the total capabilities of the symbiont interface is given in the paragraphs that follow.

5.2. OBTAINING INPUT IMAGES

5.2.1. Reading Fieldata Images (READ$)

Purpose:

Obtains an image in Fieldata from the runstream located in the run file.

Format:

```
L    A0,(EOF-return-addr,buffer-addr)
ER   READ$
```

These instructions can be generated by the procedure call:

```
R$EAD    (EOF-return-addr,buffer-addr)
```

Parameters:

EOF-return-addr	Address to which control is transferred when a control statement is encountered or end-of-file encountered during @ADD,E.
buffer-addr	Address of the input buffer into which the Fieldata image is placed.

Description:

If the input image is in ASCII, READ$ converts it to Fieldata.

Normal input image length may be up to 14 words, but images from an @ADD or @START file may be any length less than 3777_8 words.

Input images must be noncontrol statement images except for the CLIST$ control statements (see 5.5) or processor control statements (in INFOR format – see Volume 4–2.5.3). The input on auxiliary storage is not in INFOR format. It is converted by the read.

After the image is transferred to the input buffer, control is returned to the address following the READ$ request.

Upon return from an @EOF control statement (see 3.4.4.3), bits 5–0 of register A0 contain the sentinel character that appears in column 6 of the @EOF control statement and bit 35 is not set.

If the EOF return is caused by an @ADD,E control statement, H2 of register A0 is set to zero.

Upon normal return from a READ$ request, H2 of register A0 contains the number of words transferred. The meaning of any bits set in H1 of register A0 is as described in Table 5–1.

If the run is being made in the demand mode, the program is normally placed in a wait state until the READ$ request is satisfied from the demand terminal (see Section 8).

Table 5–1. Bit Settings Returned in Control Register A0 for READ$ Request

Bit Set	Description
35	Abnormal return taken because a control statement cannot be passed. Any further attempt to do READ$ request or a TREAD$ request within this program causes an error termination (unless the request is preceded by a CSF$ request with an @ADD statement).
34	Currently reading from a file or an element introduced by an @ADD control statement.
33	Set on an EOF return when at the end of an @ADD file or element and an E option was used on the @ADD control statement.
32	Set on EOF return when the user is in CLIST$ mode and a control statement specified in the user list was encountered or when a control statement that is processed by the Executive while in CLIST$ mode was encountered.
31	Image is in INFOR format (see Volume 4–2.5.3).
30	Used if bit 31 is set; indicates that more INFOR-formatted words are to be read (see Volume 4–2.5.3).
23–18	Used if a statement listed by a CLIST$ request (see 5.5) was encountered; contains the CLIST$ index value.

5.2.2. Reading ASCII Images (AREAD$)

Purpose:

Obtains an image in quarter-word ASCII from the runstream located in the run file.

Format:

```
L     A0,(EOF-return-addr,buffer-addr)
ER    AREAD$
```

These instructions can be generated by the procedure call:

```
A$READ   (EOF-return-addr,buffer-addr)
```

Parameters:

The interpretation of the parameters is identical to that for the READ$ request (see 5.2.1).

Description:

AREAD$ operation is identical to READ$ (see 5.2.1) except that normal input image length may be up to 20 words (@ADD and @START file images may be any length).

If the input image is in Fieldata, the AREAD$ request converts it to ASCII.

NOTE:

The sentinel character from an @EOF control statement is converted to Fieldata before being placed in bits 5-0 of register A0.

5.2.3. Fieldata Images - Alternate File (READA$)

Purpose:

Obtains an image in Fieldata from a user-specified file.

Format:

```
L,U   A0,pktaddr
ER    READA$
```

These two instructions may be generated by the procedure call:

```
R$EADA   pktaddr
```

Description:

Pktaddr is the address of a 3-word packet whose format is:

00	EOF-return-address	buffer-address
01	internal filename	
02		

Word 0

EOF–return–addr — The address to which control is returned when no more images exist in the file.

buffer–addr — The address of the input buffer into which the Fieldata image is placed.

Words 1 and 2

internal filename — Internal name (see 2.6.2) of the file from which the images are to be read. The filename is specified in Fieldata, left–justified and space filled.

If the input image is in ASCII, the READA$ request converts it to Fieldata.

Upon normal return from a READA$ request, register A0 contains the number of words transferred. Images may be any length less than 3777_8 words, and the caller must provide a buffer large enough to contain the largest image in the file. Normal image length is 14 words.

After the image is transferred to the input buffer, control is returned to the address following the READA$ request.

The file named in the packet must have been assigned prior to the first READA$ request and must be in SDF.

When the file is exhausted, normally by encountering an end–of–file SDF control, no image is available to transfer and the caller regains control at the EOF return address.

See 3.6.2 for the use of the @BRKPT control statement with read alternate files.

NOTE:

A READA$ request after an @BRKPT of the same file obtains the first image in the file.

5.2.4. ASCII Images – Alternate File (AREADA$)

Purpose:

Obtains an image in quarter–word ASCII from a user–specified file.

Format:

```
L,U  A0,pktaddr
ER   AREADA$
```

These two instructions may be generated by the procedure call:

```
A$READA   pktaddr
```

Description:

The interpretation of the parameters is identical to that for the READA$ request (see 5.2.3).

The AREADA$ operation is identical to READA$ request (see 5.2.3) except that the normal image length is up to 20 words.

5.2.5. Fieldata Images – Conversational Mode (TREAD$)

Purpose:

Displays the Fieldata message supplied and obtains the response in Fieldata. This request requires less overhead than an individual PRINT$ request followed by a READ$ request and should be used for demand processing.

Format:

```
L,U  A0,pktaddr
ER   TREAD$
```

Description:

Pktaddr is the address of a 2-word packet whose format is:

00	line-spacing	image-length	output-buffer-address
01	EOF-return-address		input-buffer-address

Word 0

line-spacing	The number of lines to space before displaying the message. No spacing is performed after displaying the message.
image-length	The length in words of the message.
output-buffer-address	The address of the output buffer from which the Fieldata message is obtained.

Word 1

EOF-return-address	See READ$ request (5.2.1).
input-buffer-address	The address of the input buffer into which the Fieldata image is placed.

A demand program is normally placed in a wait state until both the output and the input operations are accomplished at the terminal. An exception is made when the image transferred on the input portion is obtained from a file introduced by an @ADD control statement because there is no delay while waiting for the image to be input. In addition, the output portion is ignored when input is from an @ADD file.

Normal return is identical to that for the READ$ request (see 5.2.1).

The output portion is handled the same as in a PRINT$ request (see 5.3.1).

5.2.6. ASCII Images – Conversational Mode (ATREAD$)

Purpose:

Displays the ASCII message supplied and obtains the response in ASCII. This demand processing request is more efficient than an APRINT$ request followed by an AREAD$ request.

Format:

```
L,U  A0,pktaddr
ER   ATREAD$
```

Description:

The format of the 2-word packet and the operation of the request are identical to the TREAD$ request (see 5.2.5).

5.3. TRANSFERRING OUTPUT IMAGES

5.3.1. Printing Fieldata Images (PRINT$)

Purpose:

Places a Fieldata image into the system-defined print file.

Format:

```
L    A0,(PF line-spacing,nbr-of-words,image-addr)
ER   PRINT$
```

These two instructions may be generated by the procedure call:

```
P$RINT   (PF line-spacing,nbr-of-words,image-addr)
```

Parameters:

PF	An assembler FORM directive defined as PF FORM 12, 6, 18.
line-spacing	Number of lines to space before printing this image.
nbr-of-words	Number of data words in this image.
image-addr	Address where the Fieldata image is obtained.

Description:

The allowable values for line spacing are 0_8 to 3777_8. If the value of line spacing is greater than the number of lines remaining on the present page, the image is printed on the first printable line on the next page. If the value in line spacing is -0 (7777_8) the image is always printed on the first printable line of the next page. The first printable line on a page is defined by means of the print control margin function (see 5.4.1).

The number of words in the image is limited to the standard maximum of 22_{10} words Fieldata, unless a W print control function has been performed (see Table 5-2). This maximum is due to the 132

character width on most printers. However, if the print file is to be printed on a 0770 printer, 160 characters is the maximum width; a print control W,27 must be issued to permit creating larger images. Print images larger than the allowable width of the device are truncated at output time, unless auto-recovery is inhibited (see 5.8).

The control statement @SYM PRINT$ can be used to direct the current system-defined print file to a device other than the device indicated by device association. The queuing of the print file is held until it is closed by an @BRKPT control statement or the run is closed.

NOTE:

@SYM PRINT$ is illegal from a demand terminal.

The @BRKPT control statement is used to close and queue for printing system-defined print files. The @SYM control statement is not necessary if the user wants the file to go to the device specified by device association.

5.3.2. Printing ASCII Images (APRINT$)

Purpose:

Places a quarter-word ASCII image into the system-defined print file.

Format:

```
L    A0,(PF line-spacing,nbr-of-words,image-address)
ER   APRINT$
```

These two instructions may be generated by the procedure call:

```
A$PRINT    (PF line-spacing,nbr-of-words,image-address)
```

Parameters:

The parameters for the APRINT$ request are identical to those for the PRINT$ request (see 5.3.1).

Description:

All formats and limitations are the same as for the PRINT$ request (see 5.3.1) except a standard 132-character/line printer prints an image of 33_{10} words in quarter-word ASCII.

5.3.3. Fieldata Images - Alternate Print File (PRNTA$)

Purpose:

Places a Fieldata print image into a user-defined print file.

Format:

```
L,U  A0,pktaddr
ER   PRNTA$
```

These two instructions may be generated by the procedure call:

```
P$RNTA   pktaddr
```

Description:

Pktaddr is the address of a 3-word packet whose format is:

00	line-spacing	word-count	buffer-addr
01	internal filename		
02			

where:

The meaning of the line-spacing, word-count, and buffer-addr parameters is identical to those for the PRINT$ request (see 5.3.1), and the restrictions that apply to PRINT$ also apply to PRNTA$. Internal filename (see 2.6.2) is specified in Fieldata, left-justified and space filled.

Description:

If an alternate print or punch file has been assigned prior to the first Executive Request for the file, the Executive does not queue the file for output when it is closed. If the file has not been assigned to the run, the Executive assigns the file, and when the file is closed (@BRKPT or @FIN control statement) the file is automatically queued to the output device determined by device association. The automatic @ASG, @FREE, and @SYM of alternate symbiont output files is for batch runs only. If the alternate output file was assigned by the user, the file must be closed with an @BRKPT control statement (see 3.6.2) and queued for output with an @SYM control statement (see 3.6.3) before the run terminates for the file to be properly output.

NOTE:

An @BRKPT, followed by a print alternate to the same file begins rewriting the file.

5.3.4. ASCII Images – Alternate Print File (APRNTA$)

Purpose:

Places a quarter-word ASCII image into a user-defined print file.

Format:

```
L,U   A0,pktaddr
ER    APRNTA$
```

These two instructions may be generated by the procedure call:

```
A$PRNTA    pktaddr
```

Description:

The interpretation of parameters and operation is identical to that for the PRNTA$ request (see 5.3.3).

5.3.5. Punching Fieldata Images (PUNCH$)

Purpose:

Places a Fieldata image into the system-defined punch file.

Format:

```
L     A0,(nbr-of-words,image-addr)
ER    PUNCH$
```

These two instructions may be generated by the procedure call:

```
P$UNCH  (nbr-of-words,image-addr)
```

Parameters:

nbr-of-words	Number of words of data in this image.
image-addr	Address of the buffer where the image is obtained.

Description:

The number of words in the image must not exceed 14_{10}, unless a W punch control has been performed, and is limited by the number of characters that can be punched on the device; that is, the image is truncated at output time if it exceeds the limit of the device, unless auto-recovery is inhibited (see 5.8). If the image is to be punched in 80-column Hollerith code, the maximum image length is 14_{10}, but shorter images may be specified and are blank-filled before punching. If the image is 14_{10} words long, the last four characters must be blanks.

If the images are to be punched in column binary (see 5.4.5), an image length of 27_{10} must be used and any column that is not required for data must be zero filled.

The control statement @SYM PUNCH$ can be used to direct the current system-defined punch file to a device other than the device indicated by device association. The queuing of the punch file is held until it is closed by the @BRKPT control statement or the run is closed.

The @BRKPT control statement is used to close and queue for punching all system-defined punch files. The @SYM control statement is not necessary if the user wants the files to go to the device specified by device association.

5.3.6. Punching ASCII Images (APUNCH$)

Purpose:

Places quarter-word ASCII images into the system-defined punch file.

Format:

```
L    A0,(nbr-of-words,image-addr)
ER   APUNCH$
```

These two instructions may be generated by the procedure call:

```
A$PUNCH   (nbr-of-words,image-addr)
```

Parameter:

The parameters for the APUNCH$ request have the same meaning as the parameters for the PUNCH$ request (see 5.3.5).

Description:

The number of words in the image must not exceed 20_{10} , unless a W punch control has been performed and is limited by the number of characters that can be punched on the device; that is, the image is truncated at output time if it exceeds the limit of the device (see 5.8). If the image is to be punched in 80-column Hollerith code, the maximum image length is 20_{10}. If the image is to be punched in column binary (see 5.4.5), an image length of 27_{10} must be used and any column not punched must be zero filled.

If the user is punching column binary, there is no difference between the PUNCH$ and APUNCH$ requests.

5.3.7. Fieldata Images – Alternate Punch File (PNCHA$)

Purpose:

Places a Fieldata image into a user-defined punch file.

Format:

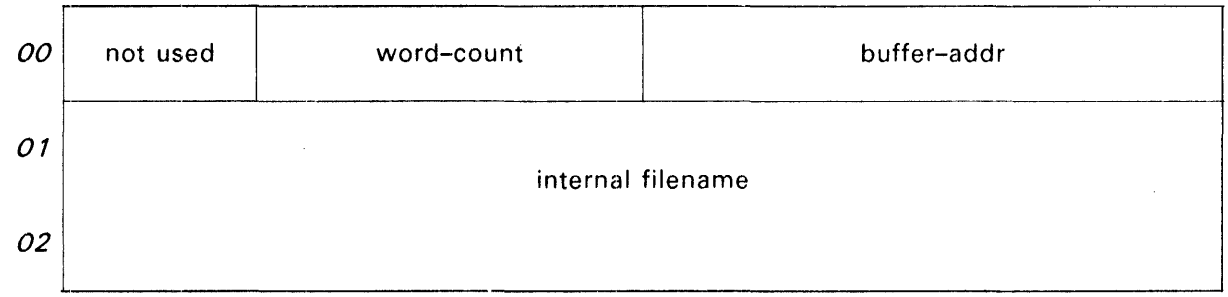
```
L,U  A0,pktaddr
ER   PNCHA$
```

These two instructions can be generated with the procedure call:

```
P$NCHA  pktaddr
```

Description:

Pktaddr is the address of a 3-word packet in the format:



Word 0

The meaning of word-count and buffer-addr are the same as for the PUNCH$ request (see 5.3.5), and all the restrictions which apply to PUNCH$ apply to PNCHA$.

Words 1 and 2

The internal filename (see 2.6.2) is specified in Fieldata, left-justified and space filled.

All rules for queuing for output of punch alternate files are the same as the rules for queuing of print alternate files (see 5.3.3).

PNCHA$ or APNCHA$ is a convenient method of building SDF files to be used later as input files in the same run or in a subsequent run. These files may be partial runstreams introduced by an @ADD control statement (see 3.10.1), complete runstreams referenced by an @START control statement (see 3.4.3), data to be referenced by READA$ or AREADA$ requests, or read directly by the user program.

5.3.8. ASCII Images – Alternate Punch File (APNCHA$)

Purpose:

Places a quarter-word ASCII punch image into a user-defined punch file.

Format:

```
L,U  A0,pktaddr
ER   APNCHA$
```

These two instructions can be generated by the procedure call:

```
A$PNCHA   pktaddr
```

Description:

The interpretation of parameters and operation is identical to that for the PNCHA$ request (see 5.3.7).

5.4. OUTPUT CONTROL FUNCTIONS

5.4.1. Fieldata Control Functions – Print File (PRTCN$)

Purpose:

Specifies Fieldata control functions to a print device routine for a print file.

Format:

```
L.  A0,(image-length,buffer-addr)
ER  PRTCN$
```

Parameters:

image-length	The length in words of the Fieldata control image. The control image must not exceed 62 words.
buffer-addr	Address of the buffer from which the Fieldata control image is obtained.

Description:

The image specified in the packet consists of one or more control functions. Each control function is in Fieldata and begins with a letter followed by subfields. A comma is the field (or subfield) separator and more than one function may be strung together separated by a period. A period is not necessary if only one function is specified.

Table 5-2 lists the print control functions and their formats.

Standard page definition, lines per inch, and width are determined by configuration parameters.

NOTE:

All values are assumed to be decimal.

Table 5-2. Print Control Functions

Control Function Format	Description
B[,lpi] ctid[,eltnam]	Except for "lpi", this is for 0770 and 0776 printers only. Use print cartridge (band) "ctid" and/or change lines-per-inch to "lpi". Non-standard cartridge-id code buffer may be found in 'eltnam' in SYS$*RUN$. For the 0768 printer, the B PRTCN$ call changes only the 8 or 6 lines-per-inch. lpi — Lines-per-inch are not decoded unless B is followed immediately by a comma. If the comma is present, lpi must be an 8 or a 6 (as desired); no leading spaces between the comma and lpi are allowed. The lpi may be the only command in this image if desired. Change to 8 lines-per-inch does not imply a page size change. Therefore, an M command should also be done. If lpi is not coded, do not code a comma after the B function. ctid — Cartridge Identification Code – this is the hexadecimal cartridge code that appears on the cartridge case. Print data following this image is printed after the operator has replaced (if necessary) the currently loaded print cartridge with this one. This field is ignored at print time if the printer is not a type 0770 or 0776. eltnam — If a special cartridge to be used is not known to the Executive, eltnam is the name of the element in SYS$*RUN$ in which this load code buffer may be found. Coding eltnam with no ctid is an error. Ctid may be the only parameter supplied if the cartridge is known to the Executive and lpi is not needed.
L,nn	Space the printer to logical line number nn–1. First logical line number is 0, (1 = 0); i.e., L, 0 and L, 1 causes logical positioning to the position 1 before the first printable line. This command causes the printer to be logically positioned so that a print command with a space count of 1 is printed on line nn.
H,options,page,text	Initiate printing page headings where: options — The available options are: N – Do not print heading X – Suppress printing page number and date page — The page number of the first page with this heading. If blank and there is no X or N option, the page numbering continues with one greater than previous page. text — The heading text (maximum of 96 characters). The text may not contain any periods. The heading is printed at the top of each succeeding page where page size is determined by the margin function described below. If a top margin of less than after heading space plus one is specified, the heading is never printed.
S,text	Special forms request for processing the print file. The text is a maximum of 48 characters. When the function is encountered for onsite printers, the text is displayed on

Table 5-2. Print Control Functions (continued)

Control Function Format	Description
	the operator's console. For batch remote devices, the text is displayed on a remote printer. The text may not contain any periods.
M,length,top,bottom	Margin setting information for readjusting page length, top and bottom margins, or space calculation mode. where: length – number of lines per page or space calculation mode top – number of blank lines used for top margin bottom – number of blank lines used for bottom margin Each parameter can be one of the following: A decimal value – "length" has a maximum of 3 digits, "top" and "bottom" have a maximum of 2 digits. An asterisk (*) – specifies the system standard value. Blank – specifies "no change" from the value currently in effect. An X in the length position causes a change in the method of calculating spacing. After an M, X spacing is not truncated by an end of page condition and top and bottom margins are zero. Any M or B command will cause this mode to be exited.
I	inhibit error recovery
A	allow error recovery The I and A functions are used to control the automatic recovery of errors for individual output files. The functions are provided to override the global on/off of auto-recovery parameters provided at system generation. Thus, even though the system may be configured to not recover automatically, the individual files may still be recovered automatically through the use of the A function. The converse holds true for the I function if auto-recovery is normally specified. These functions may be used in pairs more than once in an individual file to control automatic recovery over specific areas of output. If auto-recovery is inhibited, the onsite operator is asked via a console message if recovery should be attempted on the file (see 5.8).
D,@@CTL	The D,@@CTL allows the user to present the RSI/CCR control statements, listed in Appendix A2, to the remote symbiont complex with the same effect as if the control statement were received as input from a remote terminal. The format of the @@CTL field is identical to that of the @@ control statements as defined in Appendix A.2. For all alternate files and user assigned files where the primary output is written, this function is placed in the output file and acted upon at print time if the file is directed to an RSI-Device. This function will be ignored for all punch output.
R	Insert a logical break into print output files; used by the print symbionts when skipping forward through a print file. When the symbiont encounters the "R" function, it stops

Table 5-2. Print Control Functions (continued)

Control Function Format	Description
	skipping and requests, via the operator's console, if a reprint is desired.
W,line width	Change the maximum print width from the standard of 22_{10} words Fieldata (33_{10} words ASCII) to the value specified by the 'line width' field. Value is the width in Fieldata words. The Executive converts to ASCII words if necessary. This is only applicable to 0770 printers with feature F1533-00 and to 0776 printers.
U	Reserved for users. This option is ignored by the device handlers and is not scanned for syntax.

NOTES:

1. *H function does not take effect until spacing commands have crossed a page boundary.*
2. *S function takes immediate top of page action.*
3. *M function takes immediate top of page action.*
4. *B,lpi takes immediate top of page action, but B ctid does not cause top of page action.*
5. *B ctid,eltnam allows a user to specify a CTID and an element (absolute) name (up to 12 characters). The absolute element must be formatted as shown in Table 5-3 and must be all in one bank, preferably I-bank. If the CTID requested is not found in the element, or the element is not found, the new load is aborted.*

Table 5-3. Element Format for Cartridge ID

00	unused	"N" is the number of 3-word entries in the element or, more precisely, the number of cartridges represented in the element.
01	The Cartridge Identification Code with which the next two words are associated.	
02	The number of words in the Fieldata Load Code.	The address within the element where the Fieldata Load Code is located.
03	The number of words in the ASCII Load Code.	The address within the element where the ASCII Load Code is located.
3N+1	The Cartridge Identification Code with which the next two words are associated.	
3N+2	The number of words in the Fieldata Load Code.	The address within the element where the Fieldata Load Code is located.
3N+3	The number of words in the ASCII Load Code.	The address within the element where the ASCII Load Code is located.

An octal 0400 stop code is required after the last character to be sent. The builder of the load code buffer will be responsible for having the stop code in the proper place, even if it requires an extra word in each list.

There are two bytes required at the beginning of each load code. The first byte must be the cartridge verification code (CTID in binary), and the second must be the octal "space" code (05 in Fieldata, 040 in ASCII).

If "dualing" is desired, nine more codes are inserted between the CTID byte and the space code. These nine are 4 pairs of codes to be dualed, plus a "fall through" code to use if no match is found between the code in the print buffer and those in the load code. This latter may be another "space" code if desired. In the 4 pairs of code, the first one is "known present" code (in the LCB) to be used if the second code of the pair is found in the print line. If dualing is used, the CTID byte must have bit 7 set.

Examples of print controls:

L,5	Position the printer to logical line 4. Page ejection occurs unless the printer is at lines 1 to 4.
M,66,10,10.H,,1,MY HEADING	On 66-line paper, set top and bottom margin of 10 lines and begin a heading on the next page: page numbering is to start with 1.

5.4.2. ASCII Control Functions – Print File (APRTCN$)

Purpose:

Specifies an ASCII control function to a print device routine for a print file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   APRTCN$
```

Parameters:

image-length	The length in words of the ASCII control image. The control image must not exceed 62 words.
buffer-addr	Address of the buffer from which the ASCII control image is obtained.

Description:

The APRTCN$ request is identical to the PRTCN$ request, except that the image is in quarter-word ASCII instead of Fieldata (see 5.4.1).

5.4.3. Fieldata Control Function – Alternate Print File (PRTCA$)

Purpose:

Specifies a Fieldata control function to a print device routine for an alternate print file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   PRTCA$
```

Parameters:

image-length	The length in words of the Fieldata control image. The control image must not exceed 63 words.
buffer-addr	The address of the buffer from which the Fieldata control image is obtained.

Description:

The PRTCA$ request is identical to the PRTCN$ request (see 5.4.1), except that the first two words in the buffer specified by the buffer address in the packet must be the Fieldata internal filename of the print alternate file to which the control image is directed. The remainder of the image is the control information in Fieldata (see 5.4.1).

5.4.4. ASCII Control Functions – Alternate Print File (APRTCA$)

Purpose:

Specifies an ASCII control function to a print device routine for an alternate print file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   APRTCA$
```

Parameters:

image-length	The length in words of the ASCII control image. The control image must not exceed 63 words.
buffer-addr	The address of the buffer from which the ASCII control image is obtained.

Description:

The APRTCA$ request is identical to the PRTCN$ request (see 5.4.1), except that the first two words in the buffer specified by the buffer address in the packet must be the 12-character Fieldata internal filename of the print alternate file to which the control image is directed. The remainder of this image is the control information in quarter-word ASCII (see 5.4.1).

5.4.5. Fieldata Control Functions – Punch File (PCHCN$)

Purpose:

Specifies a Fieldata control function to a punch file routine for a punch file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   PCHCN$
```

Parameters:

image-length	The length in words of the Fieldata control image. The control image must not exceed 63 words.
buffer-addr	The address of the buffer from which the Fieldata control image is obtained.

Description:

The image specified in the packet consists of one control function. The control function is in Fieldata and begins with a letter followed by subfields. A comma is the field (or subfield) separator and a period terminates the control function string.

The punch control functions and their formats are described in Table 5-4.

Table 5-4. Punch Control Functions

Control Function Format	Description
S,text	Special forms request for processing the punch file. The text is a maximum of 54 characters. When the function is encountered for onsite punches, the text is displayed on the onsite operator's console. For batch remote devices, the text is displayed on the remote printer as soon as it is idle.
R	Insert a logical break into punch output files; used by the punch symbionts when skipping forward through a punch file. When the "R" function is encountered, the symbiont stops skipping and asks, via the operator's console, if a repunch is desired. The allowable options are to repunch "nn" cards or resume punching.
W,line width	Changes the maximum card length from 14 Fieldata words to the value specified by the "line width" field. Value is the width in Fieldata words the Executive converts to ASCII words if necessary.
C,B	Switch the mode of punching to column binary.
C,E	Switch the mode of punching to the most recently defined 80-column code.
U	Reserved for users. This option is ignored by the device handlers and is not scanned for syntax.
The following punch control functions are honored only by the symbionts interfacing with an onsite 9200/9300 and by the 0604 Punch:	
C,ASC	Switch the mode of punching to ASCII card code.
C,9000	Switch the mode of punching to EBCDIC card code.
C,1100	Switch the mode of punching to Fieldata card code.

The C,E control function is the only control function which terminates column binary mode. Other control functions may be specified within column binary mode, but only the most recent control function takes effect upon termination of column binary mode.

5.4.6. ASCII Control Function – Punch File (APCHCN$)

Purpose:

Specifies an ASCII control function to a punch device routine for a punch file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   APCHCN$
```

Parameters:

image-length	The length in words of the ASCII control image. The control image must not exceed 63 words.
buffer-addr	The address of the buffer from which the ASCII control image is obtained.

Description:

The APCHCN$ request is identical to the PCHCN$ request, except that the image is in quarter-word ASCII rather than Fieldata (see 5.4.5).

5.4.7. Fieldata Control Functions – Alternate Punch File (PCHCA$)

Purpose:

Specifies Fieldata punch control image to a punch device routine for an alternate punch file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   PCHCA$
```

Parameters:

image-length	The length in words of the Fieldata control image. The control image must not exceed 63 words.
buffer-addr	The address of the buffer from which the Fieldata control image is obtained.

Description:

The PCHCA$ request is identical to the PCHCN$ request (see 5.4.5), except that the first two words in the buffer specified by the buffer address in the packet must be the Fieldata internal filename of the alternate punch file to which the control image is directed. The remainder of the image is the control information in Fieldata (see 5.4.5).

5.4.8. ASCII Control Function – Alternate Punch File (APCHCA$)

Purpose:

Specifies an ASCII punch control image to a punch device routine for an alternate punch file.

Format:

```
L    A0,(image-length,buffer-addr)
ER   APCHCA$
```

Parameters:

image-length	The length in words of the ASCII control image. The control image must not exceed 63 words.

buffer-addr — The address of the buffer from which the ASCII control image is obtained.

Description:

The APCHCA$ request is identical to the PCHCN$ request (see 5.4.5), except that the first two words of the buffer specified by the buffer address in this packet must contain the Fieldata internal filename of the file to which the control image is directed. The remainder of the image is the control information in quarter-word ASCII (see 5.4.5).

5.5. CONTROL STATEMENT LISTING (CLIST$)

CLIST$ allows the user to define the set of control statements and register them with the Executive. On subsequent READ$ requests, the calling program is given special notification when such statements are encountered. A control statement is defined as any input image which has a master space or at symbol (@) in the first character position.

Format:

```
L    A0,list-designator
ER   CLIST$
```

Parameters:

The list designator has the following format:

0	list length	list-addr

where:

0 — Currently unused. Must be set to zero.

list length — Length of the list that is specified in H2 of the list-designator. The list header word (see below) is included in the length count. If no length is specified, 63 is assumed by the Executive.

list-addr — Address of the list of control statements the user wishes to receive.

The first entry of the list is a list header word. The format of the list header word is:

0	0	type	term

where:

0 — Currently unused. Must be zero.

type — CLIST type indicator where the types are:

1 – The user receives the image in INFOR format. The user-supplied list is in Fieldata.

2 – The user receives the images in INFOR format. The user-supplied list is in ASCII.

3 – The user receives the images in IMAGE format. The user-supplied list is in Fieldata.

4 – The user receives the images in IMAGE format. The user-supplied list is in ASCII.

5 – Terminate CLIST$ mode.

term — If set to a plus zero (+0), CLIST$ mode is terminated as soon as a READ$ request encounters a control statement that is not in one of the user lists and is not one of the control statements processed by the Executive while in CLIST$ mode (see list of statements below). After the Executive terminates CLIST$ mode, the user is given an abnormal return.

If set to 077, all control statements not in one of the user defined lists and not one of the control statements processed by the Executive while in CLIST$ mode are bypassed. A message is printed by the Executive indicating that a control statement was bypassed.

Description:

A maximum of 62 6-character alphanumeric Fieldata or ASCII control statement names may follow the list header word. The master space is assumed and is not included as part of the name. These names are left-justified and space filled.

On return from the ER CLIST$, A0 is set positive to signify normal completion, and negative if the operation was unsuccessful. If negative, S1 of A0 also contains one of the following error codes:

02 Illegal CLIST type

04 No PCT buffer available for the list

If the user-supplied list is wholly or partially outside of the user program I- or D-bank limits, the user activity is taken to error mode.

The user can receive the user-defined control statements in two formats, INFOR format (as described in Volume 4–2.5.3) and IMAGE format. IMAGE format consists of passing the control image to the user without printing it or performing syntax checks on it. When control statements are passed to the user in INFOR format, the Executive prints the control statement, and syntax checks the control statement using the rules governing processor control statements.

The user may establish two lists of user-defined control statements. One list contains names of control images that are passed to the user in IMAGE format and one list contains images that are passed to the user in INFOR format. The list that is established first by the user is searched first by the Executive. In addition, if a control statement is encountered that is not in either list and is not one of the control statements processed by the Executive while in CLIST$ mode, the list header word of the first list established governs the action taken. An existing list can be changed by the user by merely registering another list of the same type.

The Executive processes the @ADD, @EOF, @HDG, @JUMP, @LOG, @MSG, @SETC, and @TEST control statements while in CLIST$ mode if they do not appear in a user list. The only Executive control statements that are not honored as entries in a user list are @ENDX and @FIN.

When a READ$ request encounters a control statement from a user list or one of the control statements listed above, the user is given an EOF return with bit 32 of A0 set. If the user does a subsequent READ$ request and the statement encountered on the previous READ$ request was in one of the user's lists, the control image is passed to the user in the format specified by the list in which it was found. If the user does a subsequent READ$ request following the EOF return and the control statement encountered on the previous READ$ request was not in the user's lists but was one of the control statements listed above, the Executive processes the control statement. Then the next statement that the user is allowed to receive is a user-defined control statement. Any data encountered before the next user-defined control statement is encountered is ignored. Any control statements encountered before the next user-defined control statement is encountered are handled as previously described.

The user is not given an EOF return with bit 32 set when an @ADD control statement is processed by the Executive while in CLIST$ mode. The @ADD control statement is processed by the Executive, and then the first image of the ADD file is processed as the next image.

When the Executive encounters an @EOF control statement when in CLIST$ mode, the user is given an EOF return without bit 32 set in A0.

Each name in the list has an associated index value which corresponds to its position in the list (the first name is assigned index value 1). This index value is returned on a READ$ or AREAD$ request in bits 23–18 of register A0.

CLIST$ mode is terminated from the control stream by the @ENDX control statement. When the @ENDX control statement is encountered, the Executive terminates CLIST$ mode and then the user is given an EOF return with bit 32 of register A0 set and an index of 077 in bits 23–18 of register A0.

If a processor that originated from SYS$*LIB$ terminates prior to the CLIST$ mode termination, the Executive automatically terminates CLIST$ mode.

If the user program terminates prior to the CLIST$ mode termination, the Executive continues to read the control statements until the normal manner of ending the CLIST$ mode occurs. This allows PMDs even when the program has not read all the user-defined control statements. In this case the @HDG, @LOG, and @MSG control statements are not processed by the Executive. The @ADD, @EOF, @JUMP, @SETC, and @TEST control statements are processed by the Executive unless present in the list.

CLIST$ mode is terminated by the Executive before an automatic load of the PMD processor occurs.

5.6. GENERAL PACKET-DRIVEN ER INTERFACE - SYMB$

SYMB$ is a packet-driven ER that provides a means of requesting all the previously defined symbiont ER functions. It also provides the following features:

- Character transfer
- Partial READ$ transfer with the ability to retrieve or discard the remainder of the image
- Truncate capability on READ$ file requests

- Untranslated mode on READ$ requests

Format:

```
L,U  A0, packet-address
ER   SYMB$
```

The PROC call "S$YMB packet-address" generates the above calling sequence.

Parameters:

packet-address — The address of an 8- or 10-word packet. See Table 5-5, SYMB$ packet.

PROC format:

The following PROC call generates the SYMB$ packet:

S$YMBPK filename,function[,$mode_1$,...,$mode_n$] nbr-of-characters-transferred, image-address[,spacing] [nbr-of-characters,image-address]

S$YMBPK Parameters:

filename — Specifies either a standard symbiont interface file or a user alternate file.

Standard Symbiont Interface files are expressed within the PROC without quotes:

- READ$ — Used for R$ function for standard READ$ file.
- PRINT$ — Used for W$, W$R$, or SM$ functions for standard PRINT$ or TREAD$ file.
- PUNCH$ — Used for W$ or SM$ functions for standard PUNCH$ file.
- TREAD$ — Used for WR for standard TREAD$ (PRINT$ and READ$) files.

NOTE:

PRINT$ and PUNCH$ in quotes are accepted for the standard PRINT$ and PUNCH$ interfaces.

User Alternate Files are bounded by quotes when specified in the PROC and are Fieldata internal filenames.

function — describes the action to be performed on the specified file. Allowed functions are:

- R$ — 020, transfer the requested characters from filename (standard READ$ or alternate READ$ file) to the user.
- W$ — 010, transfer the user's image to the filename specified (standard PRINT$, PUNCH$, or user alternate output file).

WR 045, perform the W$ function on the standard PRINT$ file, then the R$ function on the standard READ$ file.

SM$ 042, set a mode for the specified filename as defined by the control image at "image-address"; e.g., "M,66,8,6" for PRINT$ file.

mode — provides further description of the function. Allowed modes are:

ASCII$ 01, ASCII request - for R$, return image in ASCII (convert if necessary); for W$, image at "image-addr" is in ASCII; for WR, provide ASCII on both transfers.

TRUN$ 02, truncate any image left beyond the maximum length specified in the ACW on R$ requests; i.e., discard the rest.

UNTR$ 04; untranslate mode desired on R$ requests, i.e., do not convert from ASCII to FD or vice versa.

PUALT$ 010, first call to punch alternate file for SM$ or W$ function - used to differentiate between a print alternate file and a punch alternate file.

nbr-of-characters transferred — Specifies the number of characters to transfer.

image-address — Address at which the image is located or is to be placed.

spacing — Specifies the number of lines to space prior to image output. Assumed 1 if not specified. Ignored for R$ function.

nbr-of-characters, image-address — Specifies the information for the R$ function of a WR request.

Table 5-5. SYMB$ Packet

00	filename				
01					
02	function			mode	
03	status returned	I/O status	control card index	sub-status returned	
04	character count			image-address	
05	(reserved)			final character count transferred	
06	spacing			(reserved)	EOF sentinel
07	(reserved)				
010	character count			image-address	
011	(reserved)			final character count transferred	

Words 010 and 011 - WR only

The following describes the fields in the SYMB$ packet, Table 5-5.

filename

The PROC S$YMBPK places the appropriate ER index for READ$, PRINT$, or PUNCH$ in filename; e.g., for READ$ the first two words of the packet appear as:

00	zeros	015
01	zeros	

For user alternate files, the Fieldata internal filename is placed in words 0 and 1 of the packet left-justified and space-filled.

function	As defined by the user in S$YMBPK PROC call.
mode	As defined by the user in S$YMBPK PROC call.
status returned	Status returned to the user on function completion. 00 – normal return 01 – EOF on READ$
I/O status	Contains I/O error code when SYMB$ function terminates due to an I/O error.
control card index	Contains the CLIST (see 5.5) index for the image read, if applicable.
sub–status	Further information statuses: 01 ASCII image transferred on R$. Set regardless of UNTR$ mode specified. 02 partial image transferred on R$ 010000 ↕ 0400000 } correspond to bits 30 through 35, respectively, returned in register A0 on ER READ$ requests. See Table 5–1.
character count	As defined by the user in S$YMBPK PROC call.
image–address	As defined by the user in S$YMBPK PROC call.
final character count transferred	Total number of characters transferred on each SYMB$ function call. This is set for READ$, PRINT$, and PUNCH$.
spacing	As defined by the user in S$YMBPK PROC call.
EOF sentinel	The Fieldata or ASCII character returned by an R$ in column 6 when status = 01 and substatus is not equal to 0400000.
Words 010 and 011	As previously defined, represent the R$ portion of a WR function

Description:

Because of required changes to the SDF for complete character capability, part of the internal symbiont interface remains word oriented. On W$ requests, the internal symbiont space–fills up to the nearest full word. However, on R$ requests, the "final character count transferred" is the true character count.

PRINT$ and PUNCH$ control 'W,nnn' is word oriented.

Three READ$ modes are provided via SYMB$ which are not provided via ER READ$:

1. Normal case partial transfer of long images defined by the ACW.
2. Truncate long images, discarding any images longer than the character count requested.
3. Untranslate mode.

SYMB$ requests that cause contingency notification have the SYMB$ packet address in word 1,H2 of the contingency packet.

5.7. FIELDATA AND ASCII TRANSLATION

Tables D-1 and D-2 (see Appendix D) define the software translation between ASCII and Fieldata codes as used by the language processors and the symbiont interface routine.

5.8. SYMBIONT OUTPUT FILE ERROR RECOVERY

If auto-recovery of output files is not inhibited (via configuration parameter or ER PRTCN$) and if a recoverable I/O error occurs while printing/punching a file, recovery is attempted with no intervention necessary. Recoverable I/O errors for symbiont files are defined as a status of 04, 011, or 012. Successful recovery prints or punches the message:

ERROR ENCOUNTERED WHILE OUTPUTTING FILE

at the end of the current output file.

Recovery is unsuccessful if either the I/O error is nonrecoverable or the operator terminated the file by responding to the I/O error message with a response which does not allow recovery (e.g., "G"), or, if recovery is inhibited, the operator responds "NO" to the message which asks if recovery should be attempted. If the I/O error is nonrecoverable, the message:

I/O ERROR xx. FILE TERMINATED

is printed or punched and the file output is terminated. If the operator terminated the file, the message:

I/O ERROR xx. TERMINATED BY OPERATOR

is printed or punched and the file output is terminated. In both cases xx is the I/O error code.

An image length error is defined to be an image whose length is greater than that which can be handled by the device to which the output is directed. If an image length error occurs while outputting a file and auto-recovery is not inhibited, recovery is attempted. Recovery involves verifying that the data is valid via checksum techniques, truncating the image to the length the device can handle, and then directing the output. If recovery is successful, the message:

AT LEAST 1 PRINT OR PUNCH IMAGE HAS BEEN TRUNCATED

is printed or punched at the end of the output file.

If recovery is not successful, the message:

FILE FORMAT ERROR. FILE TERMINATED

is printed or punched and the remainder of the file lost.

If auto-recovery is inhibited and an image length error occurs, the operator is asked if recovery should be attempted. If the response is no, the message:

FILE FORMAT ERROR. TERMINATED BY OPERATOR

is printed or punched and the remainder of the file lost.

If a file is too large to be printed/punched properly, the message:

FILE OVERFLOW. FILE TERMINATED

is printed or punched and the remainder of the file is lost.

Appendix A. EXEC Control Statements

A.1. SUMMARY OF CONTROL STATEMENTS

For convenience, most EXEC control statements and some processor call statements are listed here.

Transparent control statements perform the same functions as, and are listed directly following, the associated control statements. Transparent control statement format differs from standard control statement format in that the command is preceded by a double master space (@@), and use of label, while not prohibited, is meaningless.

@ACOB — Calls the ASCII COBOL language processor to translate source language instructions into relocatable binary code. The output is saved in one main relocatable output element, and possibly in several other dependent relocatable elements:

@label:ACOB,options	eltname-1,eltname-2,eltname-3,extra options

@ADD — Provides a means of inserting images into the runstream from any file currently assigned to the user or any cataloged file, provided that the file is sector-formatted in SDF or from any source element of a program file.

@label:ADD,options	filename.eltname

@ASG
@@ASG — Assigns physical facilities to a given run under the specified filename. For sector-formatted mass storage, word-addressable mass storage, and simulated word-addressable mass storage:

@label:ASG,options	filename,type/reserve/granule/maximum/ placement,pack-id-1/pack-id-2.../pack-id-n

For magnetic tape files:

@label:ASG,options	filename,type/units/log/noise/processor/ tape/format/dc,reel-1/reel-2.../reel-n,expiration-period

For all devices except sector-formatted mass storage, drum, and magnetic tape units. Used primarily for the assignment of special I/O devices and communications equipment:

@label:ASG	filename,type

@BRKPT
@@BRKPT — Enables the user to establish breakpoints in the current print and punch files (PRINT$, PUNCH$, and alternate files):

@label:BRKPT,options	generic-name/part-name
@label:BRKPT,options	generic-name,filename
@label:BRKPT,options	internal-filename
@label:BRKPT,options	filename

@CAT
@@CAT — Catalogs one or more files independent of assigned facilities.

For magnetic tapes:

@label:CAT,options	filename/read-key/write-key,type///noise/processor/tape/format/dc,reel-1/reel-2/.../reel-n

For mass storage:

@label:CAT,options	filename,type/reserve/granule/maximum/placement,pack-id-1/pack-id-2/.../pack-id-n

@CHG — Changes element name, version name, read key, write key, and mode of a file:

@label:CHG,options	name-1,name-2

@CKPT
@@CKPT — Saves the complete status of a run at a given point:

@label:CKPT,options	filename

@CLOSE — Writes two hardware EOF marks on a magnetic tape file and rewinds the tape:

@label:CLOSE,options	filename-1,filename-2,...filename-n

@COL — Permits the user to change the read mode from the system-defined standard to the read mode specified by the first parameter on the control statement. The @COL control statement is only valid when read from an onsite card reader:

@COL	xx,sentinel
@COL	command

@COPIN — Copies elements from an element file located on magnetic tape into a program file on sector-formatted mass storage:

@label:COPIN,options	name-1,name-2

@COPOUT — Copies a program file, or selected elements from a program file, located on sector-formatted mass storage onto a magnetic tape file in element file format:

@label:COPOUT,options	name-1,name-2

@COPY Copies a file or element from one file to another:

@label:COPY,options name-1,name-2,nbr-of-files

@CTS Allows the user to create, edit, and execute programs from remote terminals:

@label:CTS,options filename

@CULL Calls the CULL processor to produce an alphabetically sorted, cross-referenced listing of all symbols found and the elements and line on which they occur:

@label:CULL,options pro/scol(res),name-1,...name-n

@CYCLE Sets the maximum range of absolute F-cycle numbers to be retained for specified files which are listed in the master file directory or sets the maximum number of element cycles to be retained for a specified symbolic element:

@label:CYCLE name,n

@DATA Introduces, updates, and corrects data files from the control stream:

@label:DATA,options filename-1,filename-2,sentinel

@DELETE Drops cataloged files or marks elements in a program file as deleted:

@label:DELETE,options name-1,name-2,...,name-n

@DOC Used to produce a properly formatted printed listing of a document or to update a document:

@label:DOC,options eltname-1,eltname-2,form,form-parameters...

@ED Allows the user to conversationally edit a symbolic file or element by permitting insertion, deletion, and replacement of text:

@label:ED name-1,name-2

@ELT Introduces an element into a particular program file or makes corrections to a symbolic element in a program file from the runstream:

@label:ELT,options eltname-1,eltname-2,sentinel

@ENABLE Removes disable state from cataloged files:

@label:ENABLE filename-1,filename-2,...,filename-n

@END Notifies the system that this is the end of control stream images that are to be transferred as data by the previous @ELT,D or @DATA control statements:

@END sentinel

@ENDCL Terminates the mode established by the @COL control statement:

@ENDCL

@ENDF Terminates @FILE mode of input. Causes the file being created to be closed:

```
@ENDF
```

@ENDX When encountered by a READ$ request in CLIST$ mode, a return with a CLIST index of 77_8 occurs:

```
@ENDX
```

@EOF When encountered by a READ$ request, this control statement causes an end-of-file return:

```
@EOF                    sentinel
```

@ERS Returns to the system all mass storage granules allocated to a sector-formatted file:

```
@label:ERS              filename-1,filename-2,...,filename-n
```

@FILE Creates a file at input time from the images being read. The @FILE statement is only valid when encountered during image input:

```
@label:FILE,options     filename,type/reserve/granule/maximum,
                        pack-id-1/pack-id-2/.../pack-id-n
```

@FIN Identifies the end of a run. The @FIN control statement must appear as the last statement in all runs:

```
@FIN
```

@FIND Locates an element in a magnetic tape file (file must be in element file format) and positions the file before the element's label block:

```
@label:FIND,options     eltname
```

@FREE
@@FREE Releases the physical facilities assigned to this run under the specified filename:

```
@label:FREE,options     filename
```

@FTN Calls the ASCII FORTRAN language processor to translate source language instructions into relocatable binary code. The output is saved in one main relocatable output element, and possibly in several other dependent relocatable elements:

```
@label:FTN,options      eltname-1,eltname-2,eltname-3
```

@HDG
@@HDG Provides a means of printing a heading on successive pages of printer output along with the print file's cumulative page number and the current date:

```
@label:HDG,options      header
```

@JUMP Advances control to the specified labeled statement within the control stream:

```
@label:JUMP             label
```

@LIST — Calls the LIST processor to produce an edited dump of one or more absolute, relocatable, or source language elements:

```
@label:LIST,options      eltname-1,...,eltname-n
```

@LOG
@@LOG — Places user-specified information in the master run log:

```
@label:LOG               message
```

@MAP — Calls the @MAP processor (the Collector) to collect a specified set of relocatable elements to produce an executable program in absolute element format:

```
@label:MAP,options       eltname-1,eltname-2,eltname-3
```

@MARK — Writes two hardware EOF marks on a magnetic tape file and positions the tape between the EOF marks:

```
@label:MARK,options      filename-1,filename-2,...,filename-n
```

@MASM — Calls the MASM language processor to translate source language instructions into relocatable binary code. The output is saved in a new relocatable element:

```
@label:MASM,options      eltname-1,eltname-2,eltname-3
```

@MODE
@@MODE — Changes the mode settings initially set by a previous tape @ASG control statement. The file must be currently assigned to the run:

```
@label:MODE,options      filename,noise/MSA-trans/unit-trans/ format
```

@MOVE — Moves a magnetic tape file forward or backward over a specified number of EOF marks:

```
@label:MOVE,options      filename,n
```

@MSG
@@MSG — Used to display messages on the system console:

```
@label:MSG,options       message
```

@NUALG — Calls the NUALG language processor to translate ALGOL source language instructions into relocatable binary code. The output is saved in a new relocatable element.

```
@label:NUALG,options     eltname-1,eltname-2,eltname-3
```

@PACK — Rewrites an entire program file, removing all elements marked as deleted from the program file:

```
@label:PACK,options      filename-1,filename-2,...,filename-n
```

@PASSWD
@@PASSWD — Supplies and optionally changes the user's password.

```
@PASSWD password/new password
```

@PCH Punches program file elements into 80-column cards:

@label:PCH,options eltname,seq-char

@PDP Calls the procedure definition processor (PDP) to produce program file procedure entries suitable for use by the Assembler, COBOL, and FORTRAN language processors:

@label:PDP,options eltname-1,eltname-2

@PMD Calls the postmortem dump (PMD) processor to dump all or specified portions of a program that resides in main storage at program termination:

@label:PMD,options eltname-1,addr/loc,length,format

@label:PMD,options name-1,name-2,...,name-n

@PREP Creates an entry point table from the preambles of the nondeleted relocatable elements of a program file:

@label:PREP filename-1,filename-2,...,filename-n

@PRT Provides a listing of the master file directory items for cataloged files, information regarding temporary files, the table of contents of a program file, or the text of a symbolic element. Listings of absolute or relocatable elements must be obtained using the LIST processor:

@label:PRT,options name-1,name-2,...,name-n

@QUAL
@@QUAL Specifies a standard filename qualifier for implied use on succeeding statements involving filenames:

@label:QUAL qualifier

@REWIND Rewinds magnetic tape files back to the load point of the first reel:

@label:REWIND,options filename-1,filename-2,...,filename-n

@RSTRT Restores the run to its previous state after a checkpoint has been taken:

@label:RSTRT,priority/options run-id,acct-id,filename,ckpt-nbr,reel-nbr

@RUN Identifies the run to the Executive and provides information needed for accounting and scheduling purposes. The @RUN control statement must be the first statement of every run:

@RUN,priority/options run-id,acct-id/user-id,project-id,run-time
/deadline,pages/cards,start-time

@SETC — Stores a value in T2 of the condition word:

@label:SETC,options f/value/j

@label:SETC,options value/j

@START
@@START — Permits the user to schedule independent batch runs where the runstreams for these runs have been created and previously entered into the system:

@label:START,priority/options name,set,run-id,acct-id,project-id,run-time/deadline,pages/cards,start-time

@label:START name,set

@SYM
@@SYM — Permits the user to select a symbiont or class of symbionts to print or punch selected files:

@label:SYM,options filename,,number,device,part-name-1/part-name-2/.../part-name-n

@TEST — Tests the value of the condition word to select the particular control statements to be executed or skipped:

@label:TEST f/value/j,f/value/j

@USE
@@USE — Equates filenames so that any particular file can be referenced by more than one filename:

@label:USE internal-filename, external-filename

or

@label:USE internal-filename,internal-filename

@XQT — Initiates the execution of a program which is in absolute element notation:

@label:XQT,options eltname

A.2. SUMMARY OF DEMAND SYMBIONT CONTROL STATEMENTS

Command	Mode	Terminal	Description
@@CDI	Demand Remote Batch Inactive	DCT 1000	Turns on card input.
@@CDO	Demand Remote Batch	DCT 1000	Turns on card output.
@@CONS *keyin* @@CONS	Demand Remote Batch Inactive	all	@@CONS *keyin* allows the user to input one console keyin. @@CONS turns on console mode and each subsequent input is treated as a console keyin. @@END CONS turns off console mode.
@@CONT	Demand Remote Batch Inactive	all	Continue. Used after a break-key (interrupt of output) to signify that no special action is desired. Also used after @FIN to cause automatic resolicit of user-id/password prior to next @RUN submission or to allow another user to log-on without terminating the line.
@@CQUE	Demand	all	Circumvents input solicitation requirement. This allows several input images to be buffered in main storage before the terminal is placed in a wait condition.
@@DCT	Demand Remote Batch Inactive	TTY DCT 500	Changes terminal operating characteristics. Parameters are required.
@@END *type*	Demand	all	Terminate the special mode specified by type. *type* may be: FUL – Terminate U100/200, UTS 400 full screen input mode ESC – Terminate ESC mode INQ – Terminate INQ mode PTI – Terminate paper tape input mode CONS – Terminate CONS mode POC – Terminate UTS 400 power on confidence mode CQUE – Terminate CQUE mode CDI – Terminate DCT 1000 card read input mode

Command	Mode	Terminal	Description
			HOLD/PR/TM – Clear hold conditions as specified. null – Terminate all special modes.
@@ERLG	Demand	UTS 400	Initiates the host processor to request the error log of the UTS 400 terminal system.
@@ESC	Demand	UNISCOPE 100 UNISCOPE 200 UTS 400	Allows input to be passed to the requester unaltered from the format in which it was entered.
@@FUL	Demand Remote Batch Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Puts terminal in full-screen input mode.
@@HOLD *type/type*	Demand	all	Place a hold on type as specified as either TM for terminal message and/or PR for print output. If *type* is not specified, default to HOLD on PR and TM.
@@INQ	Demand Remote Batch Inactive	all	Directs the Executive to buffer all input to mass storage until an @@END is received. If the @@INQ is entered when the terminal is inactive, the next input should be an @RUN image. The following input is treated as remote batch input.
@@INS *n*	Demand Remote Batch Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Sets insert point at line *n*.
@@NOPR	Demand Remote Batch Inactive	DCT 1000 UNISCOPE 100 UNISCOPE 200 UTS 400	Releases assigned PAGEWRITER Printer, QTP or COP. For UNISCOPE 100/200, and UTS 400 turns off COP or QTP PAGEWRITER Printer.
@@PMOD *x*	Demand	UTS 400	Used in conjunction with the @@TCO or the @@PRNT commands. Where *x* can be one of the following options: V – transfer variable A – transfer all C – transfer changed P – print all F – print form X – print transparent The @@PMOD command is terminated or returned to Print Transparent mode (normal

Command	Mode	Terminal	Description
			mode) by entering @@TCM or @@NOPR.
@@PRNT *n*	Demand	UTS 400	Places the UTS 400 master terminal in a power-on confidence mode (POC). The @@POC mode is terminated with either an @@END POC, @@END, or an @@TERM command.
	Demand Remote Batch Inactive	DCT 1000 UNISCOPE 100 UNISCOPE 200 UTS 400	Select the printer as the output device. For UNISCOPE 100, UNISCOPE 200, and UTS 400, turns COP, QTP or PAGEWRITER Printer on. The COP or QTP number *n* must be specified for UNISCOPE 100, UNISCOPE 200, and UTS 400 unless there is only one COP or QTP.
@@PTI	Demand Remote Batch Inactive	TTY DCT 1000 DCT 500	Selects paper-tape as input. If @@PTI is entered when the terminal is inactive, the first image on paper tape should be an @RUN image. The input is then treated as a batch run.
@@PTO	Demand Remote Batch Inactive	DCT 1000	Selects paper-tape punch as output device.
@@PTP	Demand Remote Batch Inactive	DCT 1000	Enters point-to-point mode. Optional with configuration.
@@RLD	Demand Remote Batch Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Rolls screen down.
@@RLU	Demand Remote Batch Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Rolls screen up.
@@RQUE	Remote Batch	all	Stops printing batch output file but queues the file for later printing.
@@SCBY *stat-id*	Demand	UTS 400	Initiates Screen Bypass Mode (SBM). All input after entering SBM is entered as if initiating from the SBT until SBM is terminated (see @@TSBM). Where *stat-id* is the terminal-id (Terminal Descriptor ID) for the SBT.
@@SEND	Inactive	all	Prints batch output files (see Table 8-1).
@@SKIP *n*	Demand	all	Skips *n* lines of output ($n < 64$).
@@TCI *unit*, *number*	Demand Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Selects tape cassette *unit* to input *number* of blocks or until tape mark if *number* is not specified.
@@TCM	Demand	UNISCOPE 100 UNISCOPE 200 UTS 400	Tape mark to deselect tape cassette unit on input. Write tape mark and deselect tape cassette unit on output.

Command	Mode	Terminal	Description
@@TCO *unit*, *margin*	Demand Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Selects tape cassette *unit* for output. Format one image/block unless *margin* specifies the maximum number of lines to output/block.
@@TCT *unit*, *track*	Demand Inactive	UNISCOPE 100 UNISCOPE 200 UTS 400	Position tape cassette *unit* to top of *track* (1 or 2).
@@TERM	Demand Remote Batch Inactive	all	Terminates site.
@@TM *addressee/T text*	Demand	all	Send the *text* to the specified *addressee*. *run-id/R* – Send *text* to specified *run-id*. */R* must be uppercase. *user-id/U* – Send *text* to specified *user-id*. */U* must be uppercase. *site-id* – Send *text* to specified *site-id*. The *addressee* must be active.
@@TSBM	Demand	UTS 400	Exits the terminal from SBM mode (see @@SCBY). Allows the initiating terminal to return to normal demand mode while the SBT continues to perform the runstream originally initiated.
@@TTY	Demand Remote Batch Inactive	TTY DCT 500	See @@DCT.
@@X *p*			
p = I	Demand Remote Batch	all	Releases backed-up input.
p = O	Demand Remote Batch	all	Releases backed-up output.
p = T	Demand	all	Terminates executing task
p = C	Demand	all	Generates a Remote Break (RBK) contingency. If the user program has not registered to process the RBK contingency (see 4.9), action reverts to the "T" parameter if it is also present on the @@X request.
p = NONE	Demand Remote Batch	all	No parameters are equivalent to IOTC.

Appendix B. Summary of Executive Requests

ABORT$ Unconditionally terminates all activities and the run.

```
ER  ABORT$
```

ABSAD$ Allows access to downed main storage.

```
L,U  A0,pktaddr
ER   ABSAD$
```

ACSF$ Submits ASCII control statements for interpretation and processing from within an executing user program rather than from the runstream.

```
L    A0,(length,image-addr)
ER   ACSF$
```

ACT$ Activates an activity which has been previously named through the NAME$ request.

```
L    A0, activity-name
ER   ACT$
```

ADACT$ Exits from an ESI activity, releases specified buffers and activates a previously named real-time activity.

```
L    A1,name
L,U  A0,pktaddr
ER   ADACT$
```

ADED$ Dedicates an activity to a specific processor.

```
L    A0,processor mask
ER   ADED$
```

APCHCA$ Specifies an ASCII punch control image for an alternate punch file.

```
L    A0,(image-length,
     buffer-addr)
ER   APCHCA$
```

APCHCN$ Specifies an ASCII control function for a punch file.

```
L    A0,(image-length,
     buffer-addr)
ER   APCHCN$
```

APNCHA$ Places a quarter-word ASCII punch image into a user-defined punch file.

```
L,U  A0,pktaddr
ER   APNCHA$
```

APRINT$ Places a quarter-word ASCII image into the system-defined print file.

```
L    A0,(PF line-spacing,
     nbr-of-words,
     image-addr)
ER   APRINT$
```

APRNTA$ Places a quarter-word ASCII image into a user-defined print file.

```
L,U  A0,pktaddr
ER   APRNTA$
```

APRTCA$ Specifies an ASCII control function for an alternate print file.

```
L    A0,(pkt+image
     -length,pktaddr)
ER   APRTCA$
```

APRTCN$ Specifies an ASCII control function for a print file.

```
L    A0,(image-length,
     buffer-addr)
ER   APRTCN$
```

APUNCH$ Places a quarter-word ASCII image into the system-defined punch file.

```
L    A0,(nbr-of-words,
     image-addr)
ER   APUNCH$
```

AREAD$ Obtains an image in quarter-word ASCII from the runstream located in the RUN file.

```
L    A0,(EOF-return-addr,
     buffer-addr)
ER   AREAD$
```

AREADA$ Obtains an image in quarter-word ASCII from a user-specified file.

```
L,U  A0,pktaddr
ER   AREADA$
```

ATREAD$ Displays messages and receives responses in ASCII.

```
L,U  A0,pktaddr
ER   ATREAD$
```

AWAIT$ Delays further execution of the requesting activity until all specified activities have terminated.

```
L    A0,(activity-id-mask)
ER   AWAIT$
```

BANK$ Allows the user to obtain varying kinds of information pertaining to a specific bank or to change the status of a bank.

```
L    A0,parameters
ER   BANK$
```

BDSPT$ Removes a track from the available mass storage pool.

```
L    A0, (0 or pktlength,
     pktaddr)
ER   BDSPT$
```

CADD$ Returns a number of buffers to the pool.

```
L,U  A0,pktaddr
ER   CADD$
```

CEND$ Notifies the Executive that the requesting activity has completed contingency processing.

```
ER   CEND$
```

CGET$ Removes buffers from the pool only when the open chain method is employed.

```
L    A0,(nbr-of-buffers,
     pool-id)
ER   CGET$
```

CJOIN$ Expands or adds to a previously established pool by joining it to an additional pool area.

```
L,U  A0,pktaddr
ER   CJOIN$
```

CLIST$ Allows the user to define the set of control statements and register them with the Executive.

```
L    A0,list-designator
ER   CLIST$
```

CMD$ Initiates a communications handler dialing operation.

```
L,U  A0,lttaddr
ER   CMD$
```

CMH$ Initiates a communications handler hangup operation.

```
L,U  A0,lttaddr
ER   CMH$
```

CMI$ Initiates a communications handler input operation.

```
L,U  A0,lttaddr
ER   CMI$
```

CMO$ Initiates a communications handler output operation.

```
L.U  A0,lttaddr
ER   CMO$
```

CMS$ Initializes one or more line terminal groups.

```
L    A0,(lttcount,lttaddr)
ER   CMS$
```

CMSA$ Initiates a communications handler operation.

```
L,U  A0,lttaddr
ER   CMSA$
```

CMT$ Deactivates the input or output line terminals and performs various housekeeping functions associated with a line terminal group.

```
L,U  A0,lttaddr
ER   CMT$
```

COM$ Requests use of the onsite operator's console to display output messages and solicit operator input.

```
L,U  A0,pktaddr
ER   COM$
```

COND$ Transfers the program condition word to A0.

```
ER   COND$
```

CPOOL$ Establishes a pool of I/O buffers or interrupt tabling area for communications usage.

```
L,U  A0,pktaddr
ER   CPOOL$
```

CQUE$ Queues a contingency from a Common Data Bank, to be processed after control is returned from the CDB to the calling program.

```
L    A0,(error type,error code,
     contingency type)
L    A1,(auxiliary contingency
     information)
ER   CQUE$
```

CREG$ Registers a routine to handle one or more contingency types for the entire program, with the routine separate from the contingency packet.

```
L,U  A0,packet address
ER   CREG$
```

CREL$ Enables symbionts or real-time user programs to release buffer pools obtained through CPOOL$.

```
LXI,U  A0,1
ER     CREL$
         or
L,U    A0,pool-id
ER     CREL$
```

CRTN$ Removes activity from contingency state and returns control to address specified in contingency packet.

```
ER  CRTN$
```

CSF$ Submits control statements for interpretation and processing from within an executing user program rather than from the runstream.

```
L   A0,(image-wrd-
    length, image-addr)
ER  CSF$
```

C$TS Clears TSQ lock and, if activities are queued, notifies the EXEC to activate the next activity.

```
C$TS    tscell
```

C$TSA Clears a TS lock and removes an activity from a C$TSQ wait.

```
C$TSA   tscell
```

C$TSQ Clears TS lock and deactivates the calling activity.

```
C$TSQ   tscell
```

DACT$ Deactivates the calling activity which must have been previously named through the NAME$ function.

```
ER  DACT$
```

DATE$ Places in A0 and A1 the current Fieldata date and time.

```
ER  DATE$
```

EABT$ Unconditionally terminates all activities but allows error diagnostics.

```
ER  EABT$
```

EDJS$ Retrieves the jump history which is captured by the system upon the occurrence of a hardware fault interrupt.

```
L   A0,(n,buffer-addr)
ER  EDJS$
```

ERR$ Terminates the requesting activity and places it in error mode.

```
ER  ERR$
```

ERRPR$ Prints an error message for the user.

```
L,U A0,pkt-addr
ER  ERRPR$
```

EXIT$ Terminates the activity.

```
ER  EXIT$
```

FACIL$ Obtains the first nine words of the FITEM$ packet.

```
L,U A0,pktaddr
ER  FACIL$
```

FACIT$ Obtains the first 10 words of the FITEM$ packet.

```
L,U A0,pktaddr
ER  FACIT$
```

FITEM$ Provides a method to obtain a variable amount of information on file or facility assignments.

```
L   A0,(pkt-length,pktaddr)
ER  FITEM$
```

FORK$ Registers and initiates an asynchronous program activity.

```
L   A0,(parameter-word)
ER  FORK$
```

IALL$ Registers a routine to handle one or more contingency types, either for the entire program or just for the requesting activity.

```
L   A1,(extended-mask)
    [for extended
    contingencies only]
L   A0,(contingency-
    parameter)
ER  IALL$
```

IDENT$ Obtains the activity-id, name and RT priority.

```
ER  IDENT$
```

II$ Provides a means to define the activity which is to accept any unsolicited input directed to the program.

```
ER  II$
```

INFO$ Requests retrieval of run and program oriented information.

```
L   A0,(length,table-addr)
ER  INFO$
```

INT$ Asynchronously interrupts a named activity.

```
L   A0,activity-name
ER  INT$
```

IO$ Requests an operation on the I/O file indicated and returns control to the executing program without waiting for completion of the I/O operation.

```
L,U A0,pktaddr
ER  IO$
```

IOADH$ Initiates an arbitrary device I/O operation with the caller, enters a "wait for interrupt state" and controls the return to the caller.

```
L,U A0,pktaddr
ER  IOADH$
```

IOARB$ Initiates an arbitrary device I/O operation with control returned in line, as soon as the request is either listed or the operations have been initiated. (Non-1100/80 only.)

```
L,U A0,pktaddr
ER  IOARB$
```

IOAXI$ Initiates an arbitrary device operation with the referenced activity exiting function, and controls the return control to the program at the appropriate interrupt activity address specified in the request packet (non-1100/80 only).

```
L,U A0,pktaddr
ER  IOAXI$
```

IOI$ Same as IO$ except that an interrupt activity is initiated at completion of the I/O request.

```
L,U A0,pktaddr
ER  IOI$
```

IOW$ Identical to IO$ except control is not returned to the executing program until completion of the I/O operation.

```
L,U A0,pktaddr
ER  IOW$
```

IOWI$ Same as IOW$ except that an interrupt activity is initiated upon completion of the I/O operation.

```
L,U A0,pktaddr
ER  IOWI$
```

IOXI$ Requests an operation on the I/O file indicated and terminates the requesting activity. Upon completion, initiates an interrupt activity.

```
L,U A0,pktaddr
ER  IOXI$
```

LCORE$ Releases unneeded main storage.

```
L    A0,(parameter)
ER   LCORE$
```

LEVEL$ Allows a user activity to change its activity level to the lowest configured user activity level.

```
ER   LEVEL$
```

LOAD$ Loads a segment of a program.

```
L,U  A0,seg-name
L,U  A1,jump-addr
L    A2,(bank-name,
     rseg-addr)
ER   LOAD$
          or
L    A0,(0400000,seg-
     name)
L,U  A1,jump-addr
L    A2,(bank-name,
     rseg-addr)
ER   LOAD$
```

LOG$ Creates a user-formatted log entry in the SYSTEM LOG FILE.

```
L    A0,(length,address)
ER   LOG$
```

MCORE$ Obtains additional main storage.

```
L    A0,(parameter)
ER   MCORE$
```

MCT$ Retrieves information from the master configuration table (MCT) or read-updates the program area in the MCT.

```
L,U  A0,pktaddr
ER   MCT$
```

NAME$ Attaches a name to an activity so that the activity may be later referenced by an ACT$ or DACT$ activity.

```
L,U  A0,18-bit-activity-
     name
ER   NAME$
```

NRT$ Reduces an activity or program from real-time status.

```
ER   NRT$
```

OPT$ Makes available the options letters from the @XQT control statement.

```
ER   OPT$
```

PCHCA$ Specifies Fieldata punch control image for an alternate punch file.

```
L    A0,(image-length,
     buffer-addr)
ER   PCHCA$
```

PCHCN$ Specifies Fieldata control functions for a punch file.

```
L    A0,(image-length,
     buffer-addr)
ER   PCHCN$
```

PCT$ Obtains a copy of requested portions of the program's PCT in the user-specified buffer.

```
L    A0,(0,buffer-addr)
L    A1,(n,relative-addr)
ER   PCT$
          or
L    A0,(n,buffer-addr)
ER   PCT$
```

PFD$ Sets delete flag in element table item for requested element.

```
L,U  A0,pktaddr
ER   PFD$
```

PFI$ Inserts an entry in the program file's element table.

```
L,U  A0,pktaddr
ER   PFI$
```

PFS$ Searches a program file's table of contents for a given item.

```
L,U  A0,pktaddr
ER   PFS$
```

PFUWL$ Updates the next write location in a program file.

```
L,U  A0,pktaddr
L    A1,(new-address-in-
     program-file)
ER   PFUWL$
```

PFWL$ Obtains the next write location in the program file.

```
L,U  A0,pktaddr
ER   PFWL$
```

PNCHA$ Transfers a Fieldata image to a user-specified punch file.

```
L,U  A0,pktaddr
ER   PNCHA$
```

PRINT$ Transfers a Fieldata image to the system-defined print file.

```
L    A0,(PF line-spacing,
     nbr-of-words,
     image-addr)
ER   PRINT$
```

PRNTA$ Transfers a Fieldata print image into a user-defined print file.

```
L,U  A0,pktaddr
ER   PRNTA$
```

PRTCA$ Specifies Fieldata control functions for an alternate print file.

```
L    A0,(image-length,
     buffer-addr)
ER   PRTCA$
```

PRTCN$ Specifies Fieldata control functions for a print file.

```
L    A0,(image-length,
     buffer-addr)
ER   PRTCN$
```

PSR$ Sets or clears bits within the processor state register (PSR).

```
L    A0,(parameter-word)
ER   PSR$
```

PUNCH$ Transfers a Fieldata image to the system-defined punch file.

```
L    A0,(nbr-of-words,
     image-addr)
ER   PUNCH$
```

READ$ Obtains a Fieldata image from the runstream located in the READ$ file.

```
L    A0,(EOF-jump-addr,
     buffer-addr)
ER   READ$
```

READA$ Obtains a Fieldata image from a user-specified file.

```
L,U  A0,pktaddr
ER   READA$
```

ROUTE$ Dynamically alters the primary paths of communication.

```
L    A0,(mode,lttaddr)
L,U  A1,pointer
ER   ROUTE$
```

RSI$ Performs an Executive communications control routine interface function.

```
L,U  A0,pktaddr
ER   RSI$
```

RT$ Raises the program status to real-time or changes the switching priority of an activity that is already real-time.

```
L,U   A0,switching-
      priority-level
ER    RT$
        or
LN,U  A0,switching-
      priority-level
ER    RT$
```

SETBP$ Sets the 1110, 1100/40, 1100/80 Systems programmable breakpoint register.

```
L,U  A1,BDI
L    A0,(brkpt-parameter)
ER   SETBP$
```

SETC$ Dynamically sets the contents of T3 of the program condition word.

```
L,U  A0,value
ER   SETC$
```

SMU$ Allows a requestor to pass PANL directives from the 1100/80 CPU to the PANL program executing in the System Maintenance Unit (SMU).

```
L,U  A0,PACKET
ER   SMU$
```

SNAP$ Obtains a snapshot dump of selected control registers and areas of main storage.

```
S    A0,pktaddr+2
L,U  A0,pktaddr
ER   SNAP$
```

SYMB$ Specifies a symbiont function to be performed.

```
L,U  A0,pktaddr
ER   SYMB$
```

SYSBAL$ Provides the capability to turn SIP on and off from a user program and to retrieve SIP data collected.

```
L,U  A0,pktaddr
ER   SYSBAL$
```

TLBL$ Enables the user to read or write selected portions of HDR1, HDR2, EOF1, EOF2, EOV1 and EOV2 of tape label blocks. It also reads and writes user labels before and after each file.

```
L    A0,(buffer-length,
     buffer-addr)
ER   TLBL$
```

TDATE$ Places in A0 the current binary date and time.

```
ER   TDATE$
```

TFORK$ Creates a timed activity. A TFORK$ request is similar to a FORK$ request except that the new activity does not begin execution for a specified amount of time.

```
L    A0,(parameter-word)
L    A1,(wait-time-msec)
ER   TFORK$
```

TIME$ Places in A0 in binary the current time in milliseconds past midnight.

```
ER   TIME$
```

TINTL$ Causes a specified tape file to be logically rewound so that a subsequent pass may be made from the load point of the first reel.

```
L    A0,(function,pktaddr)
ER   TINTL$
```

TREAD$ Displays the Fieldata message supplied and obtains in Fieldata the response.

```
L,U  A0,pktaddr
ER   TREAD$
```

TRMRG$ Registers a Common Bank to receive control upon activity or task termination.

```
L    A0,(parameter)
ER   TRMRG$
```

TSQCL$ Deregisters TS Queuing on user program and returns TS conflict processing to normal mode.

```
ER   TSQCL$
```

TSQRG$ Notifies EXEC that user program wishes to use automatic TS Queuing on TS conflicts.

```
ER   TSQRG$
```

TSWAP$ Closes the current reel for a tape file and requests loading of the next reel.

```
L    A0,(function,pktaddr)
ER   TSWAP$
```

TWAIT$ Delays execution for a specified timed wait period.

```
L    A1,(wait-time-in-
     milliseconds)
ER   TWAIT$
```

UNLCK$ Releases the high priority of an I/O interrupt activity.

```
ER   UNLCK$
```

WAIT$ Delays execution until the I/O operation controlled by a specified I/O packet has been completed.

```
TP   pktaddr+3
ER   WAIT$
```

WALL$ Causes a wait for completion of all outstanding I/O operations for the program.

```
L,U  A0,0
ER   WALL$
```

WANY$ Delays execution of an activity until an I/O operation for that activity has completed.

```
ER   WANY$
```

Appendix C. Diagnostic Messages and Status Codes

C.1. RUNSTREAM DIAGNOSTIC MESSAGES

C.1.1. General

When a code from 1_8 to 37_8 is contained in an error message, it often points to one of the I/O problems described under type 1, error mode (EMODE) and I/O status codes (see C.3). Note that most of the messages issued by the FURPUR processor correspond to a specific I/O error and status code.

Some diagnostic messages refer to operator response keyins. Here are the usual meanings of the most common operator response keyins:

Keyin	Description
A	Try again with standard recovery
B	Return I/O status 12_8 to packet
E	Error terminate
G	Treat as unrecoverable error, since I/O device positioning appears to be good. I/O status 11_8
H	Halt the operation
I	Initiate a locked out or suspended symbiont
L	Lock out a symbiont
N	The reply is "no"
Q	Reenter a symbiont file in its appropriate queue
R	Reprint or repunch a symbiont file
S	Suspend a symbiont

T Terminate a symbiont

X Abort a run

Y The reply is "yes"

One of the three abbreviations, SI (symbolic input), RO (relocatable or absolute output), or SO (symbolic output), is frequently used to identify the element named in the corresponding specifications subfield of a processor control statement, such as @ASM, @COB, @FOR, or @MAP. For processors such as @ELT which have no RO subfield, only SI and SO are meaningful.

C.1.2. Diagnostic Messages

The runstream diagnostic messages are:

-@@IMAGE IGNORED - TRANSP CTL IN PROGRESS

A transparent control statement has been rejected because another transparent control statement from this terminal is currently being processed.

-@@COMPLETE

Normal completion of processing on a transparent control statement has taken place.

-@@ERROR (error codes)
-@@ERROR - ILLEGAL TYPE
-@@ERROR - SYNTAX

These three messages indicate that an error was encountered while processing a transparent control statement. The error codes are the same codes returned on a dynamic request of the control statement.

ABNORMAL RETURN FROM READ$

A program which expects to be called as a processor has been called with @XQT.

ACCOUNT SUA QUOTA EXCEEDED

No SUAs remain for the account specified on the @RUN control statement at initial scheduling time.

ADD ELEMENT NOT FOUND.

ADD FILE NOT ASSIGNED OR CATALOGUED

AT LEAST 1 PRINT IMAGE HAS BEEN TRUNCATED

An image length error occurred during file output and the attempted recovery was successful.

AWAIT/DEACT AMBIGUITY

The Executive determined that all activities of the run were either in an AWAIT state (ER AWAIT$) or a DEACT state (ER DACT$) and could not nor would not be activated by the run.

BAD INPUT SEQUENCE

The input control message was rejected because it conflicts with the current status of the run.

x BADLY CODED FIELD

Fill from previous field was requested and the filled field is illegal or ambiguous.

BAD RUN STATEMENT

The @RUN control statement is improperly formatted. This message is displayed only for demand runs; for batch runs the device is placed in the run search mode. If the statement was submitted from a demand terminal, it may be immediately retyped in the proper format.

BRKPT

User's breakpoint setting matched and no contingency was registered.

CANNOT SYM PU$ WHILE IN USER FILE

Caused by @SYM PUNCH$,,CP control statement when the current PUNCH$ file has been @BRKPTed to a user file.

CANNOT SYM TEMPORARY FILE

CONNECT TIME QUOTA EXCEEDED

The demand run has been open beyond the limit of its account set by QUOTA. Termination of the SITE-ID and the offending run will immediately follow.

x CYCLE SPECIFICATION IGNORED

RO and SO cycle specification is meaningless and is ignored. Does not cause error return.

DATA IGNORED -- IN CONTROL MODE

Data statements were encountered when the coarse scheduler was attempting to read control statements; that is, a program or processor was not in control of the run at the time these statements were encountered.

DBANK CANNOT BE LOADED WITH NEGATIVE BD
DBANK RELATIVE STARTING ADDRESS = OsssOOO, DBANK SIZE = OnnnOOO,
LAST ADDRESS OF USER CORE = Oxxx777.

The program cannot be loaded without causing BD to become negative. This is the result of using a SETMIN directive to specify a minimum starting address for the DBANK. See UNIVAC 1108 Processor and Storage Programmer Reference, UP-4053 (current version).

ELEMENT ADD FROM TAPE NOT ALLOWED

ELEMENT UNOBTAINABLE *xx*

The element specified on the @START control statement cannot be found. *xx* is a program file search code.

@END IGNORED - IN CONTROL MODE

An @END control statement was encountered when the coarse scheduler was attempting to read control statements; that is, the DATA OR ELT,D processor was not in control of the run at the time this statement was encountered.

@EOF ignored - IN CONTROL MODE

An @EOF control statement was encountered when the coarse scheduler was attempting to read control statements; that is, a program or processor was not in control of the run at the time this statement was encountered.

EQUIPMENT TYPE ERROR, ADD FILE

ERROR - DYNAMIC DUMPS NOT CLOSED

I/O error encountered when writing system diagnostic file (DIAG$)

ERROR ENCOUNTERED WHILE OUTPUTTING FILE

An I/O error occurred with a code of 4_8, 11_8, or 12_8 and the inhibit recovery flag had not been set by an ER PRTCN$. The message is displayed if the recovery process is successful or if the operator did not terminate the file.

ERROR LOADING PROGRAM

An error was detected in the absolute element. Bad element or the element is possibly destroyed.

NOTE:

If the error code is ERR 50 the I/O status is in A0.

ESTIMATED RUN TIME EXCEEDED

The run either exceeded the system standard or the time specified on the @RUN control statement and the T option was set.

FACILITY REJECTED *xxxxxxxxxxxx*

This message appears for a run that aborted due to a statement that cannot be honored by the system. See Table C-1 for an explanation of the 12 digit (*x...x*) octal code.

FACILITY WARNING *xxxxxxxxxxxx*

This message is a warning that the statement could cause a problem. See Table C-1 for explanation of 12 digit (*x...x*) octal code.

FILE ALREADY IN USE

@BRKPT control statement issued for a file currently being used as a symbiont file not capable of being breakpointed (@BRKPT) (e.g., an @ADD file).

x FILE CANNOT BE READ

Input file is in read-inhibited mode due to absence of read key, write-only mode is set for file, or Y option is used on the file assignment.

FILE ERROR

A FACILITY REJECT received on @ASG of a program file.

A FACILITY WARNING received on @ASG of a program file and status indicates the read key required is not supplied.

Facility status on @USE was nonzero.

Status returned from indicated file was not a program file.

The file requested on an @XQT or processor control statement could not be assigned. If the run is not in demand mode, it is terminated.

FILE FORMAT ERROR. FILE TERMINATED

An image length error occurred and the image function was set.

FILE FORMAT ERROR. TERMINATED BY OPERATOR

An image length error occurred and the operator did not try recovery.

x FILE IS TAPE

R0 and S0 may not specify tape files.

x FILE NOT AVAILABLE - STATUS: n

FILE NOT ASSIGNED: x

POSTPR$ has tried to free a file which was not assigned. The calling program has probably altered PARTBL.

FILE UNOBTAINABLE *xxxxxxxxxxxx*

The file specified on the @START control statement cannot be accessed by the Executive. *xxxxxxxxxxxx* is a 12-digit octal status code returned when the file cannot be assigned (see Table C-1).

FIRST FILENAME IS IN ERROR

First filename was not given for an @BRKPT or @SYM control statement, or an @BRKPT control statement was given for an inactive alternate file.

This message is also printed if the D option is specified on a @SYM control statement and the filename specified is not the generic name PRINT$.

ID NOT ACCEPTED

A user-id or password entered by the user is illegal.

ILLEGAL ACCOUNT

The account specified on the @RUN control statement is not in the installation's summary account file and the operator has chosen to reject it or the Executive is configured to reject all runs with undefined accounts. If the run originated from a demand device, another @RUN control statement can be entered. Batch runs are removed.

ILLEGAL ACCOUNT FILE REFERENCE

A nonprivileged run attempted to perform a privileged function on the summary account file.

nn ILLEGAL CHARACTER x

The coarse scheduler encountered an illegal character x at column nn of the above control statement.

ILLEGAL COMMON BANK BDI iii

A common bank BDI referenced by the program is invalid.

ILLEGAL CONTINUATION

Continuation of the above control statement is not allowed or the next control statement has the control character (@) in the first column. The control statement is not honored and the run is terminated (if it is not a demand run).

ILLEGAL ENTRY POINT TO A GUARANTEED ENTRY COMMON BANK

The entry point specified by the user program is not that of the guaranteed entry common bank which is initially based; or more than one guaranteed entry common bank is initially based.

ILLEGAL EQUIP ON @FILE

A file was assigned of the wrong device type. The user's read file is closed with an @FIN control statement, the run is marked as removed, and the file from the first @FILE control statement is not cataloged.

** ILLEGAL EQUIP TYPE ON FIRST FILE **

The device type on the first @FILE control statement did not specify tape equipment. This may be encountered when processing multiple @FILE control statement, or when a "T" option is found on the @FILE control statement for a mass storage file.

x ILLEGAL DEVICE

Output file is not sector-formatted or input file is neither tape nor sector-formatted.

x ILLEGAL FIELD

Field is ambiguous with option given (for example, I option specified and source output field coded).

nn ILLEGAL OPTION x

An illegal option x was encountered at column nn of the above control statement. The control statement is not honored and the run is terminated (if it is not a demand run).

ILLEGAL USER-ID

The user-id specified on the @RUN control statement is not in the installation's summary account file and the operator has chosen to reject it or the Executive is configured to reject all runs with undefined user-ids. If the run originated from a demand device, another @RUN control statement can be entered. Batch runs are removed.

IMPROPER RUNSTREAM IN FILE

The first image in the file or element specified on the @START control statement is an invalid @RUN control statement.

INTERVENING STATEMENTS SKIPPED

A conditional statement has been encountered and has caused one or more control statements to be bypassed.

INVALID SDF LABEL WORD

The input element is not SDF format or has been corrupted.

INVALID SYMBIONT NAME

@SYM file is directed to an illegal or nonexistent device or group.

I/O ERROR ENCOUNTERED

The Executive returns this message after receiving an error code while trying to read the file specified on the @START control statement.

I/O ERROR FFFFFFFFFFFF

FFFFFFFFFFFF = LIB$ or CSINTNAME$. CSINTNAME$ refers to the file specified on the preceding @XQT or @processor call card.

I/O ERROR xx. FILE TERMINATED

An I/O error occurred which is not one of the recoverable types (4_8, 11_8 and 12_8).

I/O ERROR xx WRITING DIAG$ - PMD NOT COMPLETED

I/O status xx was encountered while writing the program to DIAG$. Any PMDs are incorrect.

I/O ERROR xx. TERMINATED BY OPERATOR

An I/O error occurred and the operator terminated the file. xx is the error code.

I/U OPTION CONFLICT

Both I and U options were given on processor control statement. These are ambiguous options.

LABEL FORMAT ERROR

User tried to illegally access a labeled tape or a hardware error occurred when trying to validate a tape/disk pack label. Also, operator responded E to a request to mount a disk pack or tape.

LAST REL I – BANK ADDR GTR '0177777'

The hardware field "BS" in the PSR has 7 significant bits. This field is meaningless if set greater than 0177777. (Does not apply to 1100/80.)

L IS LEGAL FOR TAPE ONLY

The L option was specified on the @BRKPT control statement and file does not reside on tape.

MASS STORAGE OVERFLOW

Mass storage request cannot be satisfied because mass storage is not currently available.

MAX CARDS

The estimated card output has been exceeded and a system generation parameter or the C option on the @RUN control statement specified that the run be aborted if this occurred.

MASTER OR SUBMASTER KEY IN ERROR

The user has attempted to use a privileged instruction.

nn MAX NUMBER OF CHARACTERS EXCEEDED

The character at column nn of the above control statement is not a field/subfield terminator and the maximum number of characters for this subfield has been reached. The control statement is not honored and the run is terminated (if it is not a demand run).

nn MAX NUMBER OF FIELDS OR SUBFIELD EXCEEDED

The character at column nn of the above control statement is the field terminator and no more fields are permitted for that particular field. The control statement is not honored and the run is terminated (if it is not a demand run).

MAX PAGES

The estimated page output has been exceeded and a system generation parameter or the P option on the @RUN control statement specified that the run be aborted if this occurred.

MAX TIME

The estimated running time has been exceeded and a system generation parameter or the T option on the @RUN control statement specified that the run be aborted if this occurred.

MAX VOLUNTARY DELAY TIME

This message will appear on the tail sheet of any batch run which exceeds its quota of voluntary delay time. This message only appears if the run has a quota abort.

MESSAGE TOO LONG

The operator attempted to transmit a message which exceeded the input limit of 60 characters.

MISSING ACCOUNT

The account is required, but it is not specified on the @RUN control statement. If the run originated from a demand device, another @RUN control statement can be entered. Batch runs are terminated.

MISSING USER-ID

The user-id is required but it is not specified on the @RUN control statement. If the run originated from a demand device, another @RUN control statement can be entered. Batch runs are terminated.

NESTING OF @FILE STMT IS ILLEGAL

The @FILE control statement was encountered with a different filename than the preceding @FILE control statement before encountering an @ENDF control statement associated with the first @FILE control statement. The user's read file is closed with an @FIN control statement, the run is marked as removed and the file from the first @FILE control statement is not cataloged.

NO FILENAME

NO FILE SPECIFIED

Filename is not specified on an @START control statement.

NO QUOTA FOR THIS TIME/TYPE

QUOTA limits are required, but do not exist for this run type under the account specified on the @RUN control statement or quota limits do not exist for the current time under the account specified on the @RUN control statement.

NO RUN ACTIVE

This applies only to demand processing. The message appears on demand terminal if statements are entered before the @RUN control statement.

NO SPACE FOR MAJOR SAVE ON ABORT$ CONTINGENCY

No space is available in the PCT for register save when processing an ABORT$ contingency. The run is terminated.

NO SUAS REMAINING FOR ACCOUNT

No SUAs remain for the account specified on the @RUN control statement at run selection time.

NO SUAS REMAINING FOR USER-ID

No SUAs remain for the user-id specified on the @RUN control statement at run selection time.

NONZERO I/O STATUS FROM USER FILE ss

A bad I/O status was returned after an I/O request on the user's file. The file has been destroyed or there is incorrect data in the absolute element.

x NOT A PROGRAM FILE

x = SI: RO:, or SO:; file specified may not be used as a program file.

OPERATION IS ILLEGAL FOR DEMAND

@BRKPT PRINT$ or @BRKPT PRINT$/PRINT$ control statement from a demand terminal before @BRKPT PRINT$/file.

OPERATOR REMOVED RUN

The operator removed the run from the backlog by the RM keyin.

OPERATOR TERMINATED RUN BY AN E-KEYIN

The operator replied with an E to terminate the run or a demand user entered an @@X T.

OPERATOR TERMINATED RUN BY AN X-KEYIN

Operator entered an X keyin to terminate the run.

*** PARITY ERROR ***

A parity error has been detected in at least one character of the input image. The entire image is discarded. (For teletypewriter only.)

@PASSWD STATEMENT IGNORED

The user has placed an @PASSWD statement in an @ADD file.

PCT EXPANDED BEYOND SYSTEM LIMITS

The number of main storage blocks required for expansion of this run's PCT exceeds the system generation parameter PCTMAX. When a run aborts with this message, a postmortem dump of the PCT (obtained using @PMD,P) may show one of the following to be the cause:

- Excessive number of granule tables (change track granularity to position granularity)
- Excessive number of activities (check for ER FORK$ loop)
- Excessive number of files assigned (check for ER CSF$ loop)

PCT FULL, CANNOT ASSIGN ADD FILE

PCT OVERFLOW ON INITIAL LOAD

The total number of PCT blocks requested is greater than the system generation parameter PCTMAX. This message is specifically given during an initial load.

PCT/PROGRAM SIZE EXCEEDS USER CORE

Either an internal main storage bank has been downed so the program does not fit or a real-time program has started and this program is too large to fit the available main storage.

PMD NOT ALLOWED

Postmortem dump is not allowed for a system processor (called from the SYS$*LIB$) unless a Y option appeared on the @RUN control statement. If an N option appeared on the @RUN control statement, no postmortem dumps of any programs are allowed. If a W option appeared on the @RUN control statement in batch mode, postmortem dumps can be taken only if the preceding task error terminates.

PROCESSOR CALL ERROR

A program which expects to be called as a processor has been called with @XQT.

PROGRAM EXCEEDED MEMORY QUOTA

The storage requested by the run exceeds its quota limit.

PROGRAM NOT FOUND

The requested program or processor is not in the given file, LIB$, or TPF$ (depending on the statement). If the run is not demand, it is terminated.

PROGRAM TOO LARGE PROGRAM SIZE nnnnn BLKS
CORE SIZE xxxxxx BLOCKS

The program is too large to fit in the space available to user programs. The program requires nnnnn main storage blocks. A main storage block is 64_{10} (100_8) words, and the sizes nnnnn and xxxxxx are octal numbers, thus giving the sizes in octal 100s.

PUNCH FILE CANNOT BE PUNCHED: NO PUNCH DEVICES CONFIGURED

No punch device configured.

x READ ONLY OUTPUT FILE

Output file is in write inhibited mode, due to absence of write key, read-only mode set for file, or Y option used on the file assignments.

REAL-TIME NOT ALLOWED FOR THIS ACCOUNT/USER-ID

REAL-TIME-LOAD PARAMETER IGNORED

A run requesting execution of a real-time-load absolute element has an account number for which real-time is not permitted. In this case, the element is executed without real-time-load privileges.

REAL-TIME PROGRAM ATTEMPTFD PCT EXPANSION

A PCT expansion attempt for a real-time program could not be done. If the program expects PCT expansion it should use an @RUN control statement option to initialize PCT size so expansion is not attempted.

nn REQUIRED FIELD OR SUBFIELD MISSING

A field or subfield which is required on the above control statement has not been specified. The omission was detected when the field/subfield terminator or the end of the control statement was encountered at column nn. The control statement is not honored and the run is terminated (if it is not a demand run).

OPERATOR TERMINATED RUN KILLED VIA AN E-KEYIN

The operator typed an unsolicited E keyin for this run. An unsolicited E keyin for a demand run simply terminates the currently executing task.

OPERATOR TERMINATED RUN KILLED VIA AN X-KEYIN

Same as above except that the operator used the X keyin.

QUOTA ABORT

The run has exceeded one of several quota values. The exact quota parameter exceeded will appear either on the tail sheet or immediately preceding this message.

RUN REMOVED DUE TO SECURITY ERROR

The user's run has been removed because either an illegal password or no password was found in the runstream.

RUNSTREAM ANALYSIS TERMINATED

The run has been terminated because of an error condition and the remaining control statements are not processed.

SACCNT ERROR X

Internal Executive error was encountered while referencing the summary account file.

SECOND FILENAME IS IN ERROR

Caused by second filename on @BRKPT control statement not currently assigned to the user.

SECOND FILENAME NOT ASSIGNED, PUNCH$ NOT BREAKPOINTED

SECOND @FILE STATEMENT FORMAT ERROR

A syntax error was encountered on the second @FILE control statement. SECOND here implies that one or more @FILE control statements have already been processed. The user's read file is closed with an @FIN control statement, the run is marked as removed, and the file from the first @FILE control statement is not cataloged.

SECOND NAME IS ILLEGAL

Second file cannot be given on an @BRKPT of a read alternate, print alternate, or punch alternate file.

SI: CYCLE NONEXISTENT OR IN ERROR

Requested cycle of specified element does not exist or cycle field has improper format.

SI: ELEMENT NOT FOUND

Element name given cannot be found as a symbolic element in the specified program file.

SI: IMPROPER LABEL BLOCK

Source input file is tape, and tape is not positioned at the label block for requested element, probably because an @FIND has not been done.

SI IS NOT SIR TYPE SDF

The input element does not contain valid cycle information.

SI: MISSING FIELD

A field of required information (for example, element name) was not given.

SI: TAPE I/O ERROR - STATUS: n

Source input file is tape and tape error is n.

SIR EDIT ERR c r e

Line correction diagnostics produced by SIR$ in the edit mode,

where:

c – Indicates the cause of the error. A list of possible causes is shown below.

r – First four words of the range correction statement under whose control the error occurred.

e – Specifies the change correction statement that caused the error.

SEPARATOR – The separator used in the change correction statement is invalid or nonexistent.

COLUMN – The column number specified on a format 1 or 2 change correction statement is out of range, or that C > D for a format 2 change correction statement.

NO FIND – The characters given in the old data parameter of a format 3 or 4 change correction statement could not be found in the line being corrected.

NOTE:

Whenever one of the above errors occurs, the change correction statement is ignored and the line remains unchanged.

ASCII MODE – Indicates that symbolic input or output is in ASCII code, or that the user requested ASCII code. Since SIRASM cannot correct ASCII code, all range and change correction statements are ignored.

CARD COUNT < – Not enough change correction statements were provided. Those lines for which no change correction statement was provided remain unchanged.

CARD COUNT > - Too many change correction statements were provided. The excess change correction statements were ignored.

SYM FILE IS NOT CATALOGUED OR CANNOT BE FOUND IN THE DIRECTORY

TAPE IMAGE LOST*REREAD

Applies only to demand terminals – indicates that images were lost while inputting images in form II paper tape mode.

TAPE LABEL FORMAT ERROR

Tape labeling errors receive the following diagnostics:

devnam ACCESS NOT ALLOWED ON *aaa*

Where *devnam* is the logical device name of the unit and *aaa* is the reel number.

If read and write inhibit is set beyond the first label group, the following message is logged:

qualifier*filename(fff) ACCESS NOT ALLOWED ON FILE *n*

where qualifier*filename(fff) is the external filename and F-cycle of the tape file. File *n* is the tape file in which the error occurred.

Errors which occur beyond the first label group are logged to the user. The following messages are used:

qualifier*filename(fff) { EOF1 / EOF2 / HDR1 / HDR2 } LABEL NOT FOUND FOR FILE *n*

qualifier*filename(fff) FILE *n* EXPED X BLOCKS REC Y

qualifier*filename(fff) I/O ERROR Z ON { R / W } ON { TRL / HDR } LABEL ON FILE *n*

Where "X" is the number of blocks in the EOF1, EOV1, or HDR1 labels (HDR1 in reverse processing), "Y" is the number of blocks actually read when the tape mark was received. "Z" is the actual I/O error status received (other than 0), "R" and "W" indicates READ or WRITE, "TRL" indicates trailer label group, and "HDR" indicates header label group. "EXPED" refers to expected and "REC" refers to received.

The first two error log messages indicate something is missing on the tape. In both cases recovery is attempted. In the case where a label is missing the system tries to determine if any type of labels exist. If so, it tries to position accordingly. If not, it repositions after the last valid label group or at the tape mark read by the user. In the case where file *n* expected x blocks and received y, the system has picked the block count in the EOF1 or EOV1 label, compared it to the number of blocks extended and found them not equal.

In the case where the number of blocks read by the user does not correspond with the block count in the EOF1, EOF2, or HDR2, the error message is logged to the user and processing continues.

If an I/O error is detected beyond the first label group and the I/O is a read, the input buffer is checked to see if there was a partial read of a label. If there was, processing is attempted. If there was not, the file is set READ and WRITE inhibited and the user is informed that access is not allowed on that file.

TIME OUT WARNING

No activity has occurred on the line for a predefined interval. If another time interval elapses without activity, the terminal is terminated and the message TIMEOUT TERMINATION is displayed at the terminal.

TOO MANY COMMON BANKS DEFINED

The number of common banks referenced by this program is greater than the system limit.

TOO MANY SPECIFICATIONS

A processor call statement is too complex.

TOO MANY USER BANKS DEFINED

The number of user banks defined for this program is larger (250_{10}) than the maximum allowed.

UNRECOVERABLE I/O ERROR WHEN READING FILE filename

The coarse scheduler encountered an unrecoverable I/O error when searching file filename for a program or processor. If the run is not demand, it is terminated.

USER-ID DIFFERENT FROM LOG-ON

The user-id supplied on the demand @RUN statement is different from the user-id verified at log-on.

USER-ID SUA QUOTA EXCEEDED

No SUAs remain for the user-id specified on the @RUN control statement at initial scheduling time.

USER JUMP HISTORY REL. ADD.-BDI (1110 or 1100/40/80 only)

The last four jumps (1110, 1100/40) or eight jumps (1100/80) and their corresponding BDIs are printed as a result of standard action taken by the Executive on a hardware fault interrupt caused by the user. If any absolute jump address falls outside the user's limits, the address - BDI is printed as 000000 - 000000 (1110, 1100/40 only).

VOLUNTARY DELAY TIME QUOTA EXCEEDED

This message will be displayed when a demand run has exceeded its quota of voluntary delay time. Termination of the SITE-ID and the offending run will immediately follow.

WAIT LAST INPUT IGNORED

This applies only to demand terminals. It indicates that the system is not ready for further input.

WARNING IMPROPER OPTION

This is caused by @SYM,C control statement with printer symbiont name (warning only).

WARNING: MAX CONNECT TIME LIMIT IN 5 MINUTES

The demand run has only 5 minutes of its connect time quota remaining.

WARNING: MAX VOLUNTARY DELAY TIME LIMIT IN 2 MINUTES

The demand run has only 2 minutes of its voluntary delay time quota remaining.

C.2. FACILITY REQUEST STATUS CODES

If a facilities request made by one of the facilities control statements (@ASG, @MODE, @CAT, @FREE, and @USE) in error, a status word is generated in which the various bits define the error. For incorrect facilities control statements submitted in the runstream, the status word is given as part of the FACILITY REJECTED... or FACILITY WARNING... message. For control statements submitted by a CSF$ request, the status word is returned in register A0. The meanings of the possible bit settings are given in Table C-1.

Table C-1. Facility Status Bits

Bit Set	Description
35(1)	Request not accepted; check other bits for reason.
34(1)	Field error in control statement other than syntax. Option conflict (MHL,OE,or IB) or noise constant specification error. Requested hardware not currently part of the system.
33	File specified by the @ASG or @CAT control statements is already assigned, freed for the @FREE control statement specified, or not assigned for the @MODE control statement specified. For the @CAT and @MODE control statements this setting results in a FACILITY REJECTED message.
32(3)	The file was previously cataloged.
31(1)	Equipment type specified on the @ASG control statement is not compatible with cataloged type or file specified on the @MODE control statement is not magnetic tape.
30	Not used. PCT overflow.
29	That portion of the filename used as the internal name for I/O packets is not unique.
28	X option specified, file already exclusive.
27(1,2)	Incorrect read key for cataloged file.
26(1,2)	Incorrect write key for cataloged file.
25	Write key that exists in the master file directory is not specified in the @ASG control statement (file assigned in the read-only mode).
24	Read key that exists in the master file directory is not specified in the @ASG control statement (file assigned in the write-only mode).

Table C-1. Facility Status Bits (continued)

Bit Set	Description
23(1,2)	Read key specified in the @ASG control statement; none exists in the master file directory.
22(1,2)	Write key specified in the @ASG control statement; none exists in the master file directory.
21(1)	An A option was specified in the @ASG control statement and the filename cannot be found in the master file directory.
20(1)	Invalid reel number specified in the @ASG control statement for a cataloged tape file, or pack-id for cataloged removable disk file.
19(1)	Mass storage file has been rolled out.
18(1)	Request on wait status for facilities. For a tape file, this usually means a tape unit is not currently available. For a drum file, this usually is caused by an exclusive use conflict with another concurrent run.
17(1)	Option conflict for cataloged files; either the D and K options were specified; or C or U, or P, R or W in combination with C or U was specified for a file which already exists in the directory; or C was specified on an @CAT image.
16(1,3)	File assigned exclusively to another run.
15	Find was made on a cataloged file request and the file was already assigned to another run.
14(1)	File to be decataloged when no run has file assigned.
13(1)	Project-id incorrect for cataloged private file.
12(3)	Equipment is tape.
11(3)	Read-only file cataloged with an R option.
10(3)	Write-only file cataloged with a W option.
9	Equipment requested is down.
8(1)	File specified in an @ASG control statement is disabled because the links pertinent to its master file directory items have been destroyed.
7	File specified in an @ASG control statement has been disabled because the file was assigned during a file recovery.
6(1)	File specified in an @ASG control statement has been disabled because the file has been rolled out and the backup copy is unrecoverable, unless an @ENABLE command, followed by an @ASG,A command is used to retry the loading operation.

Table C-1. Facility Status Bits (continued)

Bit Set	Description
5(1)	F-cycle conflict ■ Cataloging of the requested F-cycle would force deletion of a currently assigned F-cycle. ■ F-cycle generation inhibited due to existence of +1 file. ■ F-cycle requested is not within the currently acceptable range.
4(1)	The quota limits do not allow this request. A "V" or "G" option was used and is not permitted for this account.
3	Reserved for future use.
2-0	Reserved for future use.

CODES:

(1) Request was rejected. If facility request was submitted by the runstream, this results in a FACILITY REJECTED message and termination if a batch run (results in only a FACILITY REJECTED message for demand runs). For dynamic requests through a CSF$ request, bit 35 is set in the status word returned in register A0.

(2) If the statement was submitted by a CSF$ request, the run is aborted and no status word is returned in register A0.

(3) This status bit is not returned to demand runs.

C.3. ERR MODE (EMODE) AND I/O STATUS CODES

This set of error codes is categorized as being under contingency type 12_8.

Most of these codes relate to errors that occur when users set up Executive requests (ERs). The most common user errors are improperly set up, improperly referenced, and inadvertently overwritten packets.

The ERs are categorized as follows:

Type	Mnemonic	Definition
1_8	I/O	All I/O ERs (IO$, IOWS, etc.)
2_8	SYMB	All symbiont ERs (READ$, PRINT$, etc.)
3_8	ERR$	ERR$ ER
4_8	ER	Miscellaneous ERs
5_8	CONS	COM$ ER
6_8	COM2	Communications ERs (CPOOL$, CMS$, etc.)

7_8	COMM	Communications ERs (CPOOL$, CMS$, etc.)
10_8	REP	Common bank error

Table C-2 is the full set of defined error mode codes. The following type 1 (I/O) codes 0_8 through 17_8 and 40_8 are included for the sake of completeness, even though they represent status conditions that are not necessarily errors, and do not directly force a run into error mode. Many of these codes cause the system processors to take an error exit, after passing on the code to the user in an I/O error diagnostic message.

NOTE:

Only the I/O status marked "ADI only" apply to arbitrary device interface.

Table C-2. Error Mode (EMODE) and I/O Status Codes

Error Type	Error Code Octal	Description
I/O (1)	0	The request has been completed normally, if data transfer is involved the count is given in H2 of word 4. ADI only: Interrupt status has been received. The caller must determine the validity of it.
	1	End-of-File block was detected on magnetic tape. Block read drum function was truncated by encountering an end-of-block word Block search read function was truncated by encountering an end-of-block word before the specified number of words were transferred. ADI only: The device has timed out.
	2	End of tape mark encountered on magnetic tape on a read backward from load point or on a write. No transfer takes place on the read backward. The write is done in the normal manner. Subsequent writes, barring other problems, also return an 02 status. For the mass storage EACQ$ function, a substatus is returned in S1 of word 6 which defines the cause of the rejection of the request. See 6.5.3.
	3	No find was made on a mass storage device search; the search was terminated by an end-of-block, end-of-track, or end-of-position.
	4	A nonintegral block was read from magnetic tape. The number of data characters accepted from the last word is indicated by S3 of word 4 of the packet.
	5	An attempt was made to initiate a mass storage search or read from an unassigned area or an area whose address is higher than the highest word written in the file. If the starting address is legal the read is truncated as reflected by the word count in the substatus field

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
		ADI only: PCI bit set in CCW or data format changed from one CCW to the next in a data chain.
	6	Byte interfaces only, tape unit incompatibility. The drive cannot read the tape; e.g., a NRZI tape on a UNISERVO 20 drive.
	7	A systems error has occurred. This status is created in lieu of a system halt and usually indicates an internal systems problem. ADI only: The CCW list appeared to complete in an abnormal manner–check the device status buffer for the exact reason.
	10	The area of the mass storage file being unlocked by this write or unlock request timed out in the locking list. Other requests by other activities for the area may have been honored in the interim. If the function is write, the transfer is not performed.
	11	A nonrecoverable error has occurred. The suppress recovery mode is set for magnetic tape or an answer of G was given to an error message. If the suppress recovery mode is set, the channel status words and sense bytes are returned in the packet whose address is specified in word 5 of the users I/O packet. All suppress recovery operations come back with this status.
	12	A read or write error on magnetic tape has resulted in loss of position on the unit. This code is returned for all outstanding requests at the time the answer of B was entered in response to the I/O error message. Any subsequent request is honored but no further program checkpoints are valid. For mass storage devices, this status indicates a bad spotted granule was encountered in the file or an answer of B was given to a mass storage I/O error message. ADI only: CCW quantity exceeded maximum of 254 allowed.
	13	The peripheral unit was declared down by an unsolicited operator keyin.
	14	A hardware data converter has encountered an error. Byte interface tapes only. Data converter check. ADI only: Requested path not available.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	15	ADI only: CCW data count equal to zero or invalid CCW format bits.
	16	A portion of the I/O packet specified by register A0 is not within the program limits. ADI only: EF quantity exceeded the maximum of eight allowed for a nonarbitrary device.
	17	The I/O request was not completed because of lack of EXPOOL space available to service the request.
	20	A write function was attempted on a file assigned in the read-only mode. A read function was attempted on a file assigned in the write-only mode. An ACQ$, RR$, REL$, RDL$, or UNL$ function was attempted on a file cataloged with the R option. ADI only: Invalid function specified via packet.
	21	An attempt was made to reference a filename for which no assignment has been made. ADI only: The filename specified in the ADI packet is not assigned to this run.
	22	An attempt was made to write beyond the maximum assigned space for a mass storage file. An attempt was made to expand a mass storage file beyond the maximum assigned space. A read function for a mass storage file specified an address (word 5 of the I/O packet) that is beyond the maximum assigned space. A read or write function for a word-addressable mass storage file specified a mass storage address (word 5 of the I/O packet) and a total data count. When the mass storage address is added to the total data count, the resulting ending mass storage address is greater than $2^{35}-1$ A read or write function for a sector-formatted mass storage file specified a mass storage address (word 5 of the I/O packet) that is greater than $2^{30}-1$.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
		ADI only: Reference attempted beyond the assigned file when the file is configured as a FH-432 or FH-1782 drum.
	23	The I/O packet specified by the register A0 is not within the program limits. The I/O packet specified by the register A0 is in a common bank and the request was an IO$. ADI only: The ADI packet is not within program limits.
	24	The function code is not defined for the assigned equipment type. This code also covers noncompatible fields on a set mode request. ADI only: File is not accessible via ADI.
	25	The I/O access word refers to a buffer which is wholly or partially outside of the program area. For GW$, SCR$, SCRB$, BDW$, and BDR$ functions, this error code is given if the number of access words is 0 or more than 50 or if the total word count is more than $2^{18}-1$. A BDW$ or BDR$ request has specified a BDI for a nonbased bank. In suppress recovery mode, word 5 of the I/O packet specifies a buffer which is wholly or partially outside of the program area. A R$, RB$, SCR$, SCRB$, BDW$, W$, SW$, BDR$, or a GW$ function was requested and the total word count in the ACWs is less than or equal to the noise constant. For UNISERVO VIC/VIIIC tape units, a W$, SW$, or GW$ function was attempted which would result in a zero length or EOF block being written. If an EOF was intended, use the WEF$ function. An MSA tape write in A format has the stop bit set in the first byte. An IO$ request has the access word referring to a buffer within a common bank. ADI only: Data or EF buffer not within user write enabled storage. If device being referenced is attached to an MSA, a monitor operation was encountered in the string. Otherwise access word within the packet was found completely or partially outside of program limits.
	26	ADI only: The interrupt activity starting address is out of program limits.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	27	An I/O request was made with the status word of the request packet set to octal 040 indicating the packet is already in use. ADI only: The ADI packet specified by the address in A0 is already in use by another ADI or IO request. (Most significant bit of 6-bit packet status was set at the time ADI was called.)
	30	ADI only: Device status buffer not within user write enabled storage.
	31	ADI only: CCW quantity equal to zero for function 1. A magnetic tape operation was issued with user recovery specified and entrance was not made via IOI$, IOXI$, IOWI$, or IOW$.
	32	ADI only: No channel command word list for the first function code or the CCW list is wholly or partially outside of the program area. An invalid function code was specified.
	33	A sector-formatted or word-addressable I/O request may cause the PCT to expand past its maximum. The I/O request is not initiated.
	34	A system error has occurred such that it is not possible to map the file relative address into a device relative address. An illegal absolute read or write was attempted. ADI only: Data address portion of at least one of the CCWs in the CCW list is out of program limits.
	35	Errors on read and lock, and unlock request: 1. A second read and lock (RDL$) request by an activity for a particular area. 2. An unlock (UNL$) request for an area that the activity had not previously locked. ADI only: Program controlled interrupt bit set in CCW.
	36	WAIT$ request was not immediately preceded by a test positive instruction or the test positive instruction had a nonzero h- or i-field. The operand of the test positive instruction is outside of the programs limits. The outstanding I/O request count for a program equals zero, and an activity is still waiting for completion of an I/O packet.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	40	The request is either in the process of being executed or is listed on the request queue for the particular device.
SYMB (02)	2	Attempt to read past end-of-file caused second abnormal error return from READ$ or AREAD$.
	3	I/O error (READ$, AREAD$, READA$, or AREADA$).
	4	SDF image length error.
	5	@ADD control statement error.
	6	READ$ or read alternate buffer outside of program limits.
	10	Syntax error on @ADD statement.
	11	I/O error on first read of @ADD file.
	12	Unable to assign @ADD file.
	13	Specified ADD element does not exist.
	14	Specified element cycle does not exist.
	15	Invalid equipment type for ADD file.
	16	PCT at max size – unable to complete add.
	17	I/O error on read of ADDed file.
	20	Alternate file not assigned for demand or real-time run.
	21	Cannot assign punch file or alternate print file.
	22	Type of alternate call does not match type of alternate file.
	23	Alternate packet out of program limits.
	24	Read alternate file not assigned.
	25	I/O error on first read from read alternate file.
	26	Alternate file not mass storage or tape.
	27	Cannot perform alternate ER because PCT is at maximum size.
	30	File is in use.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	31	Error in filename.
	32	SYMB$ function not defined.
	33	Error in SYMB$ packet.
	35	SYMB$ packet is in use.
	36	No images available for demand read.
	37	No buffers available for reads.
	40	PRINT$/PUNCH$ or alternate PRINT$/PUNCH$ buffer out of program limits.
	41	Maximum pages exceeded.
	42	Maximum cards exceeded.
	43	Illegal syntax on PRTCN$ or PCHCN$ control functions.
	45	I/O error while creating output file for breakpointed or alternate print/punch file.
ERR$ (03)	0	User executed ER ERR$.
ER (04)	1	ER index out range or ER for Executive only.
	2	Bad packet limits on ER.
	3	ER index within range, but not in use.
	4	Error encountered on AWAIT$ request.
	5	Bad activity number (id) specified on FORK$ request. Either out of range or already in use.
	6	Account number does not permit requested real-time priority (RT$, FORK$); or, an activity of a non-real-time load program made a positioning RT$ request at a time when all current real-time activities of the program were of the no-positioning type.
	7	Invalid queue pointer or TS cell location encountered in Test and Set Queuing.
	10	FACIL$/FACIT$/FITEM$ packet failed access word check.
	11	FACIL$/FACT/FITEM I/O error or PCT name section in error.
	17	Attempt to register MAX TIME Contingency when time remaining is at or beyond the warning time.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	20	Bad BBEOF$ packet or invalid file control table address.
	21	File not cataloged or file not sector-formatted mass storage.
	31	Illegal creation of real-time activity via FORK$ request.
	32	An activity marked as "named" cannot be found in the activity name table on a NAME$, ACTS, INT$, or DACT$ request.
	33	Illegal II$ request (II activity is already specified).
	34	Attempt to set quarter-word mode via ER PSR$ when quarter-word mode not allowed on system.
	37	Filename not assigned for tape swap.
	40	Syntax error on CSF$ image.
	41	CSF$ image length is greater than 40 words.
	42	Illegal command for CSF$.
	43	Image is outside user's program limits.
	44	Log entries for this run exceed MAX allowed by CSF$.
	45	ER ABSAD$ not allowed under the run's account number
	47	Invalid input parameter to LOAD$ (A1,H1 or A2, H1 nonzero).
	50*	I/O error encountered when loading segment.
	51	Request to load an undefined segment, or segment load table is within a write protected bank.
	52*	Invalid information in segment load table.
	53	Input parameter contains common BDI, out-of-range BDI, or an address less than the lowest address of the specified bank on ER MCORE$.
	54	Input parameter contains common BDI, out-of-range BDI, or an address less than the lowest address of the specified bank on ER LCORE$.
	55	MCORE$ request for storage not available.
	57	Attempt to release communications buffer pool with ER LCORE$.
	61	Bad packet on ER SNAP$.
	62	AO was negative on SETBP$ request.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	63	Relative breakpoint address outside user's program limits on SETBP$ request.
	64	"P" bit set in conjunction with "R" or "W" on SETBP$ request (1110 only).
	65	Bit 34 set on breakpoint parameter for SETBP$ request.
	66	Attempt to load a segment into a non-based dynamic bank, or the program control bank is a non-based dynamic bank.
	67*	An ACW in the absolute element is wholly or partially outside of the bank limits.
	70	Bad LIJ/LDJ. No bank currently assigned with index specified by LIJ/LDJ instruction, or captured P not equal to Collector defined entrance to bank.
	71	TRMRG$ was attempted from a program bank.
	76	Invalid BANK$ parameter.
	77	Processor requested on ADED$ call was unavailable at the time of the ER ADED$ or was downed by the system operator.
CONS (05)	0	Packet not within limits.
	1	Output buffer not within limits.
	2	Expected input count exceeds 50 characters.
	3	Input buffer not within limits.
	4	No valid characters in output message buffer.
COMM (06)	1	A CMD$ request has specified an invalid dial digit count or has specified a dial buffer address which is beyond the limits of the program.
	2	A CMS$ request has specified ESI completion activities with entry points in the I-bank when attached to a reentrant processor.
	3	A CMO$ request has specified: (1) an invalid pool-id for POOL mode output; (2) an invalid buffer address; or (3) a buffer which has been released.
	4	A CMO$ request for pool mode output has specified an invalid buffer character count.
	5	A CGET$ request for pool mode with 0 buffer count specified was made.
	6	A CGET$ or a CADD$ request has been made with an invalid buffer pool-id or has specified a closed-chain pool of buffers.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	11	A CMD$ request utilizing automatic dialing is only supported for the CTMC/GCS.
	12	A CMD$ request has been made with the dialing operation currently in progress.
	13	A CMD$ request has been made with an invalid BCD dial digit.
	17	A CPOOL$ or CMS$ request has failed because of insufficient user PCT space.
	20	A ROUTE$ request has been made form the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	21	A ROUTE$ request has specified an LTT address in H2 of A0 which is beyond the limits of the program.
	22	A ROUTE$ request has specified a filename which does not appear in the facility assignment section of the user's PCT.
	24	A ROUTE$ request has specified an invalid mode parameter in H1 of A0.
	25	A CADD$ request has specified an invalid buffer address or the buffer count exceeds the number of buffers in the chain.
	26	A ROUTE$ request has been made for output, but output is not defined for the alternate route.
	27	A ROUTE$ request has been made for input, but input is not defined for the alternate route.
	30	A ROUTE$ reference has requested an alternate path which has already been assigned.
	31	A ROUTE$ request has been made with a zero alternate drum address in the LTGs site table.
	32	A zero absolute drum address was computed for the alternate site table of a ROUTE$ request.
	33	A ROUTE$ request has specified a line terminal group which has not been initialized via a CMS$ request.
	34	A ROUTE$ request has specified an invalid pointer when requesting an alternate path. The pointer is greater than the number of alternates specified at system generation.
	35	A ROUTE$ request has been made for an output alternate, but that path has been downed or there is no path available.
	36	A ROUTE$ request has been made for an input alternate, but that path has been downed or there is no path available.
	40	A CMS$ request has been made with output specified for a terminal which was not set up for output at system generation time.
	41	A CMS$ request has been made with input specified for a terminal which was not set up for input at system generation.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	42	An ER CMS$ reference has been made with an illegal C/SP initialization parameters.
	46	A CMI$ request has been made for pool mode input on an LT that has not been initialized for pool mode input.
		C/SP rejected initialization data.
	47	A CPOOL$ request has been made with an invalid activity name.
	50	A CMS$ request has been made with an invalid tabling-id.
COMM (07)	1	A CMS$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	2	A CMS$ request has specified an LTT address in H2 of A0 which is beyond the limits of the program.
	3	The initialization request specified an LT table which is currently in use or has been initialized but has never been terminated, or a terminal which is inoperable at this time.
	4	A CMO$, CMI$, or CMSA$ request has been made with the path being downed by the system.
	5	An invalid value was specified for the input completion activity usage code within the requesting packet for a CMS$ request.
	6	The filename for an initialization request does not appear in the facility assignment section of the user's PCT.
	7	The end-of-input activity code in the packet for an initialization request is invalid.
	10	A CMI$, CMO$, or CMSA$ request has been made with the LTG having been deactivated due to an ESI contingency error, or no ESI contingency was specified for an ESI contingency.
	11	The control group address specified in a packet to be used for pool mode operation is invalid, or the pool is closed chained and shared usage was attempted.
	12	Pool mode operation has been specified, but no buffers presently exist in the pool specified as the one to be used.
	13	The input completion activity address specified within the packet for a CMS$ request is beyond the limits of the program.
	14	The output usage code specified within the packet for an initialization request is invalid.
	15	A CMS$ request was made specifying output pool mode with an invalid pool-id or with closed mode.
	16	A CMT$ request has specified an LT group which has been terminated previously or has never been initialized.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	17	A CMT$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	20	A CMT$ request has specified an LTT address in H2 of A0 which is beyond the limits of the program.
	21	A request for an input operation has specified a filename for which input is not permissible. This indicates that no input facility for the LT group was defined at system generation.
	22	An invalid value was specified for the input completion usage code within the requesting packet for a CMI$ request.
	23	The number of characters specified for a single mode input or output operation exceeds the maximum permissible value established at system generation, or equals zero.
	24	CMI$, CMO$, or CMSA$ request has specified an LTT address in H2 of A0 which is beyond the limits of the program.
	25	The character specified count for an input operation on the CTMC subsystem was not a multiple of six.
	26	An input operation has specified a character count and buffer starting address such that a portion of the input buffer exists beyond the limits of the program.
	27	An input or output operation has specified pool mode operation for a filename associated with equipment which uses the ISI method of buffering. Buffer swapping for ISI devices is not permitted because the hardware operates on data by blocks rather than as a continuous input stream.
	30	The filename for a CMI$, CMO$, or CMSA$ request does not appear in the facility assignment section of the user's PCT.
	31	A request for output operation has specified a filename for which output is not permissible. This indicates that no output facility for the LT group was defined at system generation.
	32	An invalid value was specified for the output completion usage code within the requesting packet for a CMO$ request.
	33	A CMO$ request specifying pool mode operation has been made at the same time from more than one activity using the same LTT.
	34	An output operation has specified a character count and a buffer starting address such that a portion of the output buffer exists beyond the limits of the program.
	35	An initialization request has specified a filename associated with equipment which is not handled by the communications complex.
	36	A CPOOL$ request specified an illegal pool area because: (1) the pool size equals zero words; (2) pool area is split between user program's D- and I-banks; or (3) pool area is beyond the limits

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
		of user's program. The main storage area specified by a CPOOL$ request is located in the program's I-bank. Pool buffering using the I-bank is not permitted since a program should be positioned in main storage to allow the hardware storage overlap feature for ESI data transfers to reduce the number of main storage accesses from three to two for each transfer.
	40	A CPOOL$ request has specified an illegal buffer size because: 1. character count equal zero; or 2. character count exceeds system's specified maximum.
	41	A CPOOL$ request has specified a pool area which overlaps that of a previous request which has yet to be released.
	42	A CMI$, CMO$, CMSA$, CPOOL$, or CREL$ request to the communications handler has been executed from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	43	The single mode of buffering has specified an input buffer which is in the I-bank. Communications buffers are not permitted in the I-bank because real-time programs are positioned in main storage to allow the D-bank to utilize the storage overlap feature for each character transfer.
	44	A CMS$ request for pool mode operations and buffer mode conflicts with the subsystem mode specified on the SGS configuration card.
	45	The address within the packet specified as the output completion activity was not valid for a CMS$ request.
	46	A CMI$ or CMO$, request has been made requesting use of an LT group by the communication handler before that LT group has been initialized via a CMS$ request.
	50	A STOP AUTO POLL request was made on a line not currently doing Auto polling or a second Auto poll/recycle Auto poll request was made on a line currently doing Auto poll/recycle Auto poll.
	53	A CMS$ request has specified an LTT which exists in the real-time program's I-bank. Real-time programs are expected to utilize the hardware feature of main storage overlap by being divided into appropriate I- and D-bank portions.
	54	A CMD$ or CMH$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	55	The dial usage code specified within the packet for a CMD$ request is invalid.
	56	A CMD$ or CMH$ request has specified an LTG which has been terminated previously or has never been initialized.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	57	The address within the packet specified as the dial completion activity address was not valid for a CMD$ request.
	60	ESI contingency error for contingency types 1 through 4 from the ESI activity.
	61	ESI contingency error for ACT$ or ADACT$ requests from the ESI completion activity.
	62	ESI contingency error for CADD$ or ADACT$ requests from the ESI completion activity.
	63	ESI contingency error for illegal ER from the ESI completion activity.
	64	An ESI completion activity for an ESI contingency activity has exceeded the maximum amount of execution time set at system generation.
	65	A CMT$ request has specified a filename which does not appear in the facility assignment section of the user's PCT.
	66	A CMD$ or CMH$ request has specified a filename which does not appear in the facility assignment section of the user's PCT.
	67	A CMD$ or CMH$ request has specified a filename for which no dialing or hang-up capability exists.
	70	A CREL$ or CJOIN$ request has specified an invalid pool name.
	71	A CREL$ request has specified an invalid release code.
	72	A CREL$ request has specified a buffer pool which is not assigned to the requestor.
	73	A CREL$ request has specified a pool to be released which is marked as being used for input or output.
	74	Neither input nor output mode has been specified in the packet for an initialization request.
	75	A CMD$ request has been made with no path available or with a path marked as down by the system.
	76	A CMS$ request has been made with no output path available or the path has been downed by the system.
	77	A CMS$ request has been made with no input path available or the path has been downed by the system.
REP (10)	17	I/O error loading bank
	21	Attempt to change size of a write-protected bank via the ERs, MCORE$ or LCORE$.

Table C-2. Error Mode (EMODE) and I/O Status Codes (continued)

Error Type	Error Code Octal	Description
	22	The active program's main storage requirement exceeded total user storage.
	23	Activity was terminated in the process of effecting a type 2 common bank reload (terminate activities with the common bank based).
	24	Attempt to reference an unbased common bank using ER MCORE$ or ER LCORE$.
	25	Attempt to reload a common bank with a bankname which is not in the common bankname table.
	26	Stall reload request aborted, privileged mode was not set or bank's file was not properly assigned (correct keys).
	27	Bad reload request type or packet was not within limits.
	30	Invalid BDI on BANK$ request.
	31	Bank not based on a dynamic to static BANK$ change request.
	32	Attempt to make a utility PSR reference on a 2-bank machine 1108/06, 1110/20 Systems.
		Reload requested but reload bank not based.
	33	Stall reload request errored because a stall request is already queued.
	34	Absolute element specified for common bank reload could not be found.
	35	Old format absolute element not reloadable.
	36	I/O error was encountered when reading the header table or bankname table while attempting a common bank reload.
	37	Bad bankname. Bankname not found in bankname table.
	40	The absolute element identified for common bank reload of a standalone bank contained more than one bank of nonzero length.
	41	A "stall" type of common bank reload was requested which specified a stall time greater than the maximum allowed.
	42	A type-2 common bank reload was specified for an non-based common bank.
	43	Attempt to base a bank which previously failed load.
	44	Based common bank has been reloaded and there were activities on the Test and Set queue.

* *Program file load errors type 04, codes 50, 52, and 67 will cause the appropriate error message to be displayed and also a word of auxiliary status whose format is as follows.*

reserved for internal use by Exec	major code — a unique 9-bit code for each error	minor code — a 9-bit secondary code dependent on the major code

The minor code may contain a lower level subroutine's major code.

The following list contains definitions of the major code portion of the auxiliary status. Where applicable, the minor codes associated with each major code are also listed.

Major Code	Minor Code	Description
001		Invalid Bank Descriptor Index (BDI) specified in common block ACW
002		All Bank Control Packets (BCP) from BLP have been seen
003		End sentinel found in Load Control Group (LCG)
004		Zero-fill ACW found in LCG
005		Common block ACW found in LCG
006		End sentinel not found when expected
	005	Common block ACW found in LCG
010		Zero-fill ACW found outside of bank
011		Load control group makes no sense
	005	Common block ACW found in LCG
012		Segment LCG ACW outside of bank
	003	End sentinel found in LCG
	005	Common block ACW found in LCG
013		Attempt to load a segment in a write protect bank
	003	End sentinel found in LCG
	005	Common block ACW found in LCG
014		An Load Control Entry (LCE) for a simple segment is bad
	005	Common block ACW found in LCG
015		Attempt to load a segment in a common bank
	003	End sentinel found in LCG
	005	Common block ACW found in LCG
016		Attempt to load segment in nonreferenced bank
	003	End sentinel found in LCG
	005	Common block ACW found in LCG
017		I/O error received on LCG read
	status	I/O status from bad LCT read
020		I/O error received on text read
	status	I/O status from bad text read
022		Bad BDI for RSEG load
023		BDI in range, but not in use for RSEG load
024		Bad BDI taken from extended Segment Load Table (SLT)
025		In range, but not in use BDT for main segment load
026		Bad segment load table extension link
027		Could not successfully assign common bank file
030		Bad BDI taken from SLT extension
031		I/O error on TIP absolute element validity check
032		Failure of TIP absolute element validity check

C.4. CSF$ EXECUTIVE REQUEST STATUS CODES

C.4.1. Facility Request Status Codes (@CAT, @ASG, @FREE, @MODE, @USE)

The meanings of the bits of the status code returned in register A0 for dynamic facility requests are described in Table C-1.

C.4.2. @SYM and @BRKPT Status Codes

The status codes returned in register A0 for dynamic @SYM or @BRKPT control statements are given in Table C-3, with the equivalent diagnostic that would have been generated if the control statement had been submitted in the runstream (see C.1 for explanation of the diagnostics).

Table C-3. @SYM and @BRKPT Status Codes

Status Code (Octal)	Equivalent Diagnostic
1	First field required for L option BRKPT
2	No file specification for breakpoint
3	Excess fields specified on breakpoint
4	Max BRKPTS exceeded for PRINT$/PUNCH$
5	BRKPT PRINT$ is illegal for demand
6	File specified on BRKPT not assigned
7	Alternate file being breakpointed has not been referenced
10	I/O status writing EOF
11	File being breakpointed to is in use
12	Illegal syntax for L option breakpoint
13	No filename specified on SYM image
14	SYM of PRINT$ illegal from demand mode
15	SYM before BRKPT on user defined file illegal
16	Filename error
17	Fac status *n* on attempted ASG
20	@SYM of temporary file is illegal

Table C-3. @SYM and @BRKPT Status Codes (continued)

Status Code (Octal)	Equivalent Diagnostic
21	D option illegal for user file
22	C option illegal for non-punch type file
23	Device name specified is not configured
24	SYM of system file is illegal
25	File status bad on lookup
26	Read of symed (@SYM) file illegal
27	Drop of SYMed file illegal
30	SYM of file in +1 state illegal
31	No punch devices configured
32	File marked to be dropped - SYM illegal
33	PRINT$/PUNCH$ illegal as second filename
34	No main item found for SYMed file
35	Number of copies specified contains illegal character
36	Illegal user-id specified
37	No user-id configured
40	PCT expansion error
41	BRKPT tp write-inhibited file not allowed

C.4.3. @ADD Status Codes

The status codes returned in register A0 for dynamic @ADDs are defined in Table C-2 under Type 2 error diagnostics for SYMB.

C.4.4. @START Diagnostics and Status Codes

On return from a CSF$ @START control statement, register A0 contains the codes described in Table C-4.

Table C-4. @START Status Code

Status Codes (Octal)	Description
0	Request Processed Normally
1	Request Rejected Due To Improper Runstream In File
2	Request Rejected Due To File Unobtainable
3	Request Rejected Due To Element Obtainable
4	Request Rejected Due To Filename Not Specified
5	I/O Error Encountered
6	File Not Cataloged On Mass Storage

C.5. GUARD MODE AND UNDEFINED SEQ. STATUS (1110, 1100/40)

Since the guard mode fault and the undefined sequence fault are the only faults that generate a status word, and since the undefined sequence fault (other than residency) is similar to a guard mode fault, the two faults are logically combined so that both appear as a guard mode fault to the user. The following describes the status bits returned in the user's contingency packet (see 4.9.4):

Word 1 (S3)

002	Illegal attempt to attach to an EXEC bank or table length violation (LIJ/LBJ/LDJ error)
004	Privileged instruction violation
010	Storage limits violation
020	Control register violation
040	Interrupt lockout violation (interrupts locked out for more than 100 μsec.)

The following are error codes for guard mode/undefined sequence interrupts returned in the user's contingency packet:

Word 0 (S2)

001	Privileged instruction violation
002	Storage limits/write protect violation
003	Control register violation
004	Interrupt lockout violation
006	E-Bit violation
007	Table length violation
010	Entry point violation

C.6. GUARD MODE AND ADDRESSING EXCEPTION STATUS FOR 1100/80

The addressing exception fault and the guard mode fault both appear as a guard mode fault to the user. Both of these faults result in the hardware providing a full word of status information which is presented to the user as part of the contingency information.

The following describes the status returned in the users contingency packet (see 4.9.4):

Word 1

The hardware status word associated with either the addressing exception or guard mode interrupt. For status word format, see 1100/80 System Processor and Storage Programmer Reference, UP-8492 (current version) or UP-8604 (current version).

The error codes for guard mode/addressing exception interrupts are the same as for the Series 1110 System with the following additions:

Word 0 (S2)

011	Use Count Overflow on New BD or Use Count Underflow on Old BD

Appendix D. Conversion Tables

D.1. INTRODUCTION

This appendix provides a series of conversion tables as an aid to the programmer. The following conversion tables are printed:

Table D-1	Fieldata-to-ASCII Conversion
Table D-2	ASCII-to-Fieldata Conversion
Table D-3	Cursor/SOE Coordinates (UNISCOPE 100/200)
Table D-4	Binary/Hexadecimal Conversion
Table D-5	Octal/Decimal Conversion
Table D-6	80 Column Card Code, Symbol, XS-3, Fieldata, BCD Conversion

D.2. ASCII AND FIELDATA CONVERSION TABLES

Codes, which also represent collating sequence, are given in octal in Tables D-1 and D-2.

ASCII codes from 00_8 to 37_8 are for communication, format, and separator control characters. These are not converted into Fieldata.

The ASCII symbols represented by codes 40_8 to 137_8 are converted into the identical Fieldata symbols, except that the quotation marks symbol is converted into a lozenge, the circumflex is converted into a delta, and the underscore is converted into a not equal sign.

There are no remaining unique Fieldata symbols into which to convert the balance of the ASCII symbols, represented by codes 140_8 to 177_8, so these codes are "folded" over codes 100_8 to 137_8 (by clearing bit 5, which amounts to subtracting 40_8). This means that ASCII codes 101_8 (A) and 141_8 (a), for example are both transiated as if they were codes 101_8 (converted to Fieldata 06_8 for A). These "folded" codes are shown boxed in (Table D-2).

Although ASCII is presently a 7-bit code, it may eventually be extended to eight bits to allow for additional controls and special graphic characters, including possibly whole alternate alphabets. On a 36-bit machine, each ASCII code is stored within a 9-bit quarter word.

Table D-1. Fieldata-to-ASCII Conversion

Fieldata		ASCII		Fieldata		ASCII	
Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol
00	@	100	@	40	)	51	)
01	[	133	[	41	- (minus)	55	- (minus)
02	]	135	]	42	+	53	+
03	#	43	#	43	<	74	<
04	Δ	136	^	44	=	75	=
05	SP	40	SP	45	>	76	>
06	A	101	A	46	&	46	&
07	B	102	B	47	$	44	$
10	C	103	C	50	*	52	*
11	D	104	D	51	(	50	(
12	E	105	E	52	%	45	%
13	F	106	F	53	: (colon)	72	: (colon)
14	G	107	G	54	?	77	?
15	H	110	H	55	!	41	!
16	I	111	I	56	, (comma)	54	, (comma)
17	J	112	J	57	\	134	\
20	K	113	K	60	0	60	0
21	L	114	L	61	1	61	1
22	M	115	M	62	2	62	2
23	N	116	N	63	3	63	3
24	O	117	O	64	4	64	4
25	P	120	P	65	5	65	5
26	Q	121	Q	66	6	66	6
27	R	122	R	67	7	67	7
30	S	123	S	70	8	70	8
31	T	124	T	71	9	71	9
32	U	125	U	72	' (apos)	47	' (apos)
33	V	126	V	73	;	73	;
34	W	127	W	74	/	57	/
35	X	130	X	75	. (period)	56	. (period)
36	Y	131	Y	76	¤	42	"
37	Z	132	Z	77	≠	137	_

Table D-2. ASCII-to-Fieldata Conversion

ASCII		Fieldata		ASCII		Fieldata	
Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol
40	SP	05	SP	106	F	13	F
41	!	55	!	107	G	14	G
42	"	76	¤	110	H	15	H
43	#	03	#	111	I	16	I
44	$	47	$	112	J	17	J
45	%	52	%	113	K	20	K
46	&	46	&	114	L	21	L
47	' (apos)	72	' (apos)	115	M	22	M
50	(	51	(	116	N	23	N
51	)	40	)	117	O	24	O
52	*	50	*	120	P	25	P
53	+	42	+	121	Q	26	Q
54	, (comma)	56	, (comma)	122	R	27	R
55	- (minus)	41	- (minus)	123	S	30	S
56	. (period)	75	. (period)	124	T	31	T
57	/	74	/	125	U	32	U
60	0	60	0	126	V	33	V
61	1	61	1	127	W	34	W
62	2	62	2	130	X	35	X
63	3	63	3	131	Y	36	Y
64	4	64	4	132	Z	37	Z
65	5	65	5	133	[	01	[
66	6	66	6	134	\	57	\
67	7	67	7	135	]	02	]
70	8	70	8	136	^	04	Δ
71	9	71	9	137	—	77	≠
72	: (colon)	53	: (colon)	140	`	00	@
73	;	73	;	141	a(1)	06	A(2)
74	<	43	<	through	through	through	through
75	=	44	=	172	z	37	Z
76	>	45	>	173	{	01	[
77	?	54	?	174	:	57	\
100	@	00	@	175	}	02	]
101	A	06	A	176	~	04	Δ
102	B	07	B	177	DEL	77	≠
103	C	10	C				
104	D	11	D				
105	E	12	E				

NOTES:

1. *Lowercase alphabet.*
2. *Uppercase alphabet.*

D.3. SPECIAL CHARACTERS IN ASCII

SP designates space, which is normally nonprinting.

DEL designates delete, and has a code of all 1 bits.

This code obliterates any unwanted previous character on paper tape or other nonerasable medium.

Names of some special characters in ASCII are:		Some additional standardized names of interest:	
"	Quotation marks	#	Number sign
^	Circumflex	&	Ampersand
_	Underline	'	Apostrophe
`	Grave accent	*	Asterisk
{	Opening brace	>	Greater than sign
|	Vertical line	@	At sign
}	Closing brace	[	Opening bracket
~	Tilde	\	Reverse slant

Definitions of the 32 ASCII control characters, codes 00_8 to 37_8 are:

00	NUL	Null – all zeros character which may serve as time fill	
01	SOH	Start of heading	
02	STX	Start of text	
03	ETX	End of text	
04	EOT	End of transmission	
05	ENQ	Enquiry – "Who Are You?"	
06	ACK	Acknowledge – "Yes"	
07	BEL	Bell – human attention required	
10	BS	Backspace	format effectors for printing or punching
11	HT	Horizontal tabulation	
12	LF	Line feed	
13	VT	Vertical tabulation	
14	FF	Form feed	
15	CR	Carriage return	
16	SO	Shift out – nonstandard code follows	

17	SI	Shift in – return to standard code
20	DLE	Data link escape – change limited data communications controls
21	DC1	Device controls for turning on or off auxiliary devices
22	DC2	
23	DC3	
24	DC4	
25	NAK	Negative acknowledge – "No"
26	SYN	Synchronous idle – from which to achieve synchronism
27	ETB	End of transmission block – relates to physical communication blocks
30	CAN	Cancel previous data
31	EM	End of medium – end of used or desired portion of information
32	SUB	Substitute character for one in error
33	ESC	Escape – for code extension – change some character interpretations
34	FS	File separator
35	GS	Group separator
36	RS	Record separator
37	US	Unit separator
177	DEL	Medium fill

These information separators are ordered in descending hierarchy. They are followed by ASCII 40_8 (space), which can also be considered as a word separator.

D.4. UNISCOPE 100/200, UTS 400 DISPLAY TERMINAL

Table D-3 provides the Fieldata equivalents of the ASCII representations of various UNISCOPE 100/200, UTS 400 functions and characters.

Table D-3. Cursor/SOE Coordinates

x/y Coordinate	ASCII		Fieldata		x/y Coordinate	ASCII		Fieldata	
	Character	Value (Octal)	Numerics	Value (Octal)		Character	Value (Octal)	Numerics	Value (Octal)
1		40	1	6061	41	H	110	41	6461
2	!	41	2	6062	42	!	111	41	6462
3	"	42	3	6063	43	J	112	43	6463
4	#	43	4	6064	44	K	113	44	6464
5	$	44	5	6065	45	L	114	45	6465
6	%	45	6	6066	46	M	115	46	6466
7	&	46	7	6067	47	N	116	47	6467
8	' (apos)	47	8	6070	48	O	117	48	6470
9	(	50	9	6071	49	P	120	49	6471
10	)	51	10	6160	50	Q	121	50	6560
11	*	52	11	6161	51	R	122	51	6561
12	+	53	12	6162	52	S	123	52	6562
13	, (comma)	54	13	6163	53	T	124	53	6563
14	- (minus)	55	14	6164	54	U	125	54	6564
15	. (period)	56	15	6165	55	V	126	55	6565
16	/	57	16	6166	56	W	127	56	6566
17	0	60	17	6167	57	X	130	57	6567
18	1	61	18	6170	58	Y	131	58	6570
19	2	62	19	6171	59	Z	132	59	6571
20	3	63	20	6260	60	[	133	60	6660
21	4	64	21	6261	61	\	134	61	6661
22	5	65	22	6262	62	]	135	62	6662
23	6	66	23	6263	63	^	136	63	6663
24	7	67	24	6264	64	_	137	64	6664
25	8	70	25	6265	65		140	65	6665
26	9	71	26	6266	66	a	141	66	6666
27	: (colon)	72	27	6267	67	b	142	67	6667
28	;	73	28	6270	68	c	143	68	6670
29	<	74	29	6271	69	d	144	69	6671
30	=	75	30	6360	70	e	145	70	6760
31	>	76	31	6361	71	f	146	71	6761
32	?	77	32	6362	72	g	147	72	6762
33	@	100	33	6363	73	h	150	73	6763
34	A	101	34	6364	74	i	151	74	6764
35	B	102	35	6365	75	j	152	75	6765
36	C	103	36	6366	76	k	153	76	6766
37	D	104	37	6367	77	l	154	77	6767
38	E	105	38	6370	78	m	155	78	6770
39	F	106	39	6371	79	n	156	79	6771
40	G	107	40	6460	80	o	157	80	7060

D.5. BINARY/HEXADECIMAL CONVERSION TABLE

Table D-4 provides binary/hexadecimal equivalents.

Table D-4. Binary/Hexadecimal Conversion

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

D.6. OCTAL/DECIMAL CONVERSION TABLE

The table of octal/decimal equivalents (Table D-5) provides a rapid means of converting from octal to decimal and vice versa. The range of the table is 0000 - 4095_{10} to 0000 - 7777_8.

To convert a decimal number greater than 4095 to its octal equivalent, reduce the number to 4095 or less by subtracting sufficient multiples of 4096. Convert this residue to octal by means of the table, and add 10000_8 for each multiple of 4096 which had been subtracted.

To convert an octal number greater than 7777 to the decimal equivalent, reduce the number to 7777 or less by subtracting sufficient multiples of 10000. Convert this residue to octal by means of the table, and add 4096_{10} for each multiple of 10000 which had been subtracted.

Table D-5. Octal/Decimal Conversion

OCTAL 0000 to 0777	DECIMAL 0000 to 0511

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

OCTAL 1000 to 1777	DECIMAL 0512 to 1023

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

Table D–5. Octal/Decimal Conversion (continued)

OCTAL 2000 to 2777	DECIMAL 1024 to 1535

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

OCTAL 3000 to 3777	DECIMAL 1536 to 2047

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

Table D-5. Octal/Decimal Conversion (continued)

OCTAL 4000 to 4777	DECIMAL 2048 to 2559

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

OCTAL 5000 to 5777	DECIMAL 2560 to 3071

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

Table D-5. Octal/Decimal Conversion (continued)

OCTAL 6000 to 6777	DECIMAL 3072 to 3583

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

OCTAL 7000 to 7777	DECIMAL 3584 to 4095

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

D.7. 80-COLUMN CARD CODE CONVERSION TABLE

Table D-6 cross references Fieldata 80-column card codes to the Fieldata, ASCII, and XS-3 codes and symbols. XS-3 is the code used for data transfers to and from on-site 1004's and REM-1 remote terminals.

Table D-6. Fieldata 80-Column Card Code Conversion Table

Fieldata 80-Column Card Code	Fieldata Symbol	Fieldata Octal	ASCII Octal from Fieldata Cards	ASCII Symbol from Fieldata Cards	XS-3 Octal
12-1	A	06	101	A	24
12-2	B	07	102	B	25
12-3	C	10	103	C	26
12-4	D	11	104	D	27
12-5	E	12	105	E	30
12-6	F	13	106	F	31
12-7	G	14	107	G	32
12-8	H	15	110	H	33
12-9	I	16	111	I	34
11-1	J	17	112	J	44
11-2	K	20	113	K	45
11-3	L	21	114	L	46
11-4	M	22	115	M	47
11-5	N	23	116	N	50
11-6	O	24	117	O	51
11-7	P	25	120	P	52
11-8	Q	26	121	Q	53
11-9	R	27	122	R	54
0-2	S	30	123	S	65
0-3	T	31	124	T	66
0-4	U	32	125	U	67
0-5	V	33	126	V	70
0-6	W	34	127	W	71
0-7	X	35	130	X	72
0-8	Y	36	131	Y	73
0-9	Z	37	132	Z	74
0	0	60	60	0	03
1	1	61	61	1	04
2	2	62	62	2	05
3	3	63	63	3	06
4	4	64	64	4	07
5	5	65	65	5	10
6	6	66	66	6	11
7	7	67	67	7	12
8	8	70	70	8	13
9	9	71	71	9	14
12	+	42	53	+	20
11	- (minus)	41	55	- (minus)	02
12-0	?	54	77	?	23
11-0	!	55	41	!	43
0-1	/	74	57	/	64

Table D-6. Fieldata 80-Column Card Code Conversion Table (continued)

Fieldata 80-Column Card Code	Fieldata Symbol	Fieldata Octal	ASCII Octal from Fieldata Cards	ASCII Symbol from Fieldata Cards	XS-3 Octal
2-8	&	46	46	&	63
3-8	=	44	75	=	35
4-8	' (apos)	72	47	' (apos)	56
5-8	: (colon)	53	72	: (colon)	21
6-8	>	45	76	>	76
7-8	@	00	100	@	40
12-3-8	. (period)	75	56	. (period)	22
12-4-8	)	40	51	)	75
12-5-8	[	01	133	[	17
12-6-8	<	43	74	<	36
12-7-8	#	03	43	#	37
11-3-8	$	47	44	$	42
11-4-8	*	50	52	*	41
11-5-8	]	02	135	]	01
11-6-8	;	73	73	;	16
11-7-8	Δ	04	136		57
0-2-8	≠	77	137	- -	60
0-3-8	, (comma)	56	54	, (comma)	62
0-4-8	(	51	50	(	61
0-5-8	%	52	45	%	55
0-6-8	\	57	134	\	15
0-7-8	¤	76	42	"	77
b	(space)	05	40	(space)	00

D.8. EBCDIC/ASCII 80-COLUMN CARD CODE CONVERSION TABLE

Table D-7 cross references 80 column ASCII card codes to ASCII, Fieldata, and symbols.

Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion

80-Column Card Code	ASCII Octal from ASCII Cards	ASCII Symbol from ASCII Cards	Fieldata Octal from ASCII Cards	Fieldata Symbol from ASCII Cards	EBCDIC Hexidecimal
12-0-9-8-1	000	ƀ	05	ƀ	00
12-9-1	001	ƀ	05	ƀ	01
12-9-2	002	ƀ	05	ƀ	02
12-9-3	003	ƀ	05	ƀ	03
9-7	004	ƀ	05	ƀ	37
0-9-8-5	005	ƀ	05	ƀ	2D
0-9-8-6	006	ƀ	05	ƀ	2E
0-9-8-7	007	ƀ	05	ƀ	2F
11-9-6	010	ƀ	05	ƀ	16
12-9-5	011	ƀ	05	ƀ	05
0-9-5	012	ƀ	05	ƀ	25
12-9-8-3	013	ƀ	05	ƀ	OB
12-9-8-4	014	ƀ	05	ƀ	OC
12-9-8-5	015	ƀ	05	ƀ	OD
12-9-8-6	016	ƀ	05	ƀ	OE
12-9-8-7	017	ƀ	05	ƀ	OF
12-9-8-1	020	ƀ	05	ƀ	10
11-9-1	021	ƀ	05	ƀ	11
11-9-2	022	ƀ	05	ƀ	12
11-9-3	023	ƀ	05	ƀ	13
9-8-4	024	ƀ	05	ƀ	3C
9-8-5	025	ƀ	05	ƀ	3D
9-2	026	ƀ	05	ƀ	32
0-9-6	027	ƀ	05	ƀ	26
11-9-8	030	ƀ	05	ƀ	18
11-9-8-1	031	ƀ	05	ƀ	19
9-8-7	032	ƀ	05	ƀ	3F
0-9-7	033	ƀ	05	ƀ	27
11-9-8-4	034	ƀ	05	ƀ	1C
11-9-8-5	035	ƀ	05	ƀ	1D
11-9-8-6	036	ƀ	05	ƀ	1E
11-9-8-7	037	ƀ	05	ƀ	1F
Δ	040	ƀ	05	ƀ	40
12-8-7	041	!	055	!	4F
8-7	042	"	076	¤	7F
8-3	043	#	030	#	7B
11-8-3	044	$	047	$	5B
0-8-4	045	%	052	%	6C
12	046	&	046	&	5D
8-5	047	'	072	'	7D
12-8-5	050	(	051	(	4D
11-8-5	051	)	040	)	5D

Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion (continued)

80-Column Card Code	ASCII Octal from ASCII Cards	ASCII Symbol from ASCII Cards	Fieldata Octal from ASCII Cards	Fieldata Symbol from ASCII Cards	EBCDIC Hexidecimal
11-8-4	052	*	050	*	5C
12-8-6	053	+	042	+	4E
0-8-3	054	,	056	,	6B
11	055	-	041	-	60
12-8-3	056	.	075	.	4B
0-1	057	/	074	/	61
0	060	0	060	0	F0
1	061	1	061	1	F1
2	062	2	062	2	F2
3	063	3	063	3	F3
4	064	4	064	4	F4
5	065	5	065	5	F5
6	066	6	066	6	F6
7	067	7	067	7	F7
8	070	8	070	8	F8
9	071	9	071	9	F9
8-2	072	:	053	:	7A
11-8-6	073	;	073	;	5E
12-8-4	074	<	043	<	4C
8-6	075	=	044	=	7E
0-8-6	076	>	045	>	6E
0-8-7	077	?	054	?	6F
8-4	0100	@	0	@	7C
12-1	0101	A	06	A	C1
12-2	0102	B	07	B	C2
12-3	0103	C	010	C	C3
12-4	0104	D	011	D	C4
12-5	0105	E	012	E	C5
12-6	0106	F	013	F	C6
12-7	0107	G	014	G	C7
12-8	0110	H	015	H	C8
12-9	0111	I	016	I	C9
11-1	0112	J	017	J	D1
11-2	0113	K	020	K	D2
11-3	0114	L	021	L	D3
11-4	0115	M	022	M	D4
11-5	0116	N	023	N	D5
11-6	0117	O	024	O	D6
11-7	0120	P	025	P	D7
11-8	0121	Q	026	Q	D8
11-9	0122	R	027	R	D9
0-2	0123	S	030	S	E2
0-3	0124	T	031	T	E3
0-4	0125	U	032	U	E4
0-5	0126	V	033	V	E5
0-6	0127	W	034	W	E6
0-7	0130	X	035	X	E7
0-8	0131	Y	036	Y	E8

Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion (continued)

80-Column Card Code	ASCII Octal from ASCII Cards	ASCII Symbol from ASCII Cards	Fieldata Octal from ASCII Cards	Fieldata Symbol from ASCII Cards	EBCDIC Hexidecimal
0-9	0132	Z	037	Z	E9
12-8-2	0133	[	01	[	4A
0-8-2	0134	\	057	\	EO
11-8-2	0135	]	02	]	5A
11-8-7	0136	^	04	Δ	5F
0-8-5	0137	←	077	≠	6D
8-1	0140	`	0	@	79
12-0-1	0141	a	06	A	81
12-0-2	0142	b	07	B	82
12-0-3	0143	c	010	C	83
12-0-4	0144	d	011	D	84
12-0-5	0145	e	012	E	85
12-0-6	0146	f	013	F	86
12-0-7	0147	g	014	G	87
12-0-8	0150	h	015	H	88
12-0-9	0151	i	016	I	89
12-11-1	0152	j	017	J	91
12-11-2	0153	k	020	K	92
12-11-3	0154	l	021	L	93
12-11-4	0155	m	022	M	94
12-11-5	0156	n	023	N	95
12-11-6	0157	o	024	O	96
12-11-7	0160	p	025	P	97
12-11-8	0161	q	026	Q	98
12-11-9	0162	r	027	R	99
11-0-2	0163	s	030	S	A2
11-0-3	0164	t	031	T	A3
11-0-4	0165	u	032	U	A4
11-0-5	0166	v	033	V	A5
11-0-6	0167	w	034	W	A6
11-0-7	0170	x	035	X	A7
11-0-8	0171	y	036	Y	A8
11-0-9	0172	z	037	Z	A9
12-0	0173	{	010	[	CO
12-11	0174	|	057	\	6A
11-0	0175	}	020	]	DO
11-0-1	0176	~	040	Δ	A1
12-9-7	0177	ъ	077	≠	07
11-0-9-8-1	0200	ъ	05	ъ	20
0-9-1	0201	ъ	05	ъ	21
0-9-2	0202	ъ	05	ъ	22
0-9-3	0203	ъ	05	ъ	23
0-9-4	0204	ъ	05	ъ	24
11-9-5	0205	ъ	05	ъ	25
12-9-6	0206	ъ	05	ъ	06
11-9-7	0207	ъ	05	ъ	17
0-9-8	0210	ъ	05	ъ	28
0-9-8-1	0211	ъ	05	ъ	29

Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion (continued)

80-Column Card Code	ASCII Octal from ASCII Cards	ASCII Symbol from ASCII Cards	Fieldata Octal from ASCII Cards	Fieldata Symbol from ASCII Cards	EBCDIC Hexidecimal
0-9-8-2	0212	b	05	b	2A
0-9-8-3	0213	b	05	b	2B
0-9-8-4	0214	b	05	b	2C
12-9-8-1	0215	b	05	b	09
12-9-8-2	0216	b	05	b	0A
11-9-8-3	0217	b	05	b	1B
12-11-0-9-8-1	0220	b	05	b	30
9-1	0221	b	05	b	31
11-9-8-2	0222	b	05	b	1A
9-3	0223	b	05	b	33
9-4	0224	b	05	b	34
9-5	0225	b	05	b	35
9-6	0226	b	05	b	36
12-9-8	0227	b	05	b	08
9-8	0230	b	05	b	38
9-8-1	0231	b	05	b	39
9-8-2	0232	b	05	b	3A
9-8-3	0233	b	05	b	3B
12-9-4	0234	b	05	b	04
11-9-4	0235	b	05	b	14
9-8-6	0236	b	05	b	3E
11-0-9-1	0237	b	05	b	E1
12-0-9-1	0240	b	05	b	41
12-0-9-2	0241	b	05	b	42
12-0-9-3	0242	b	05	b	43
12-0-9-4	0243	b	05	b	44
12-0-9-5	0246	b	05	b	45
12-0-9-6	0245	b	05	b	46
12-0-9-7	0246	b	05	b	47
12-0-9-8	0247	b	05	b	48
12-8-1	0250	b	05	b	49
12-11-9-1	0251	b	05	b	51
12-11-9-2	0252	b	05	b	52
12-11-9-3	0253	b	05	b	53
12-11-9-4	0254	b	05	b	54
12-11-9-5	0225	b	05	b	55
12-11-9-6	0226	b	05	b	56
12-11-9-7	0257	b	05	b	57
12-11-9-8	0260	b	05	b	58
11-8-1	0261	b	05	b	59
11-0-9-2	0262	b	05	b	62
11-0-9-3	0263	b	05	b	63
11-0-9-4	0264	b	05	b	64
11-0-9-5	0265	b	05	b	65
11-0-9-6	0266	b	05	b	66
11-0-9-7	0267	b	05	b	67
11-0-9-8	0270	b	05	b	68
0-8-1	0271	b	05	b	69

Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion (continued)

80-Column Card Code	ASCII Octal from ASCII Cards	ASCII Symbol from ASCII Cards	Fieldata Octal from ASCII Cards	Fieldata Symbol from ASCII Cards	EBCDIC Hexidecimal
12-11-0	0272	ƀ	05	ƀ	70
12-11-0-9-1	0273	ƀ	05	ƀ	71
12-11-0-9-2	0274	ƀ	05	ƀ	72
12-11-0-9-3	0275	ƀ	05	ƀ	73
12-11-0-9-4	0276	ƀ	05	ƀ	74
12-11-0-9-5	0277	ƀ	05	ƀ	75
12-11-0-9-6	0300	ƀ	05	ƀ	76
12-11-0-9-7	0301	ƀ	05	ƀ	77
12-11-0-9-8	0302	ƀ	05	ƀ	78
12-0-8-1	0303	ƀ	05	ƀ	80
12-0-8-2	0304	ƀ	05	ƀ	8A
12-0-8-3	0305	ƀ	05	ƀ	8B
12-0-8-4	0306	ƀ	05	ƀ	8C
12-0-8-5	0307	ƀ	05	ƀ	8D
12-0-8-6	0310	ƀ	05	ƀ	8E
12-0-8-7	0311	ƀ	05	ƀ	8F
12-11-8-1	0312	ƀ	05	ƀ	90
12-11-8-2	0313	ƀ	05	ƀ	9A
12-11-8-3	0314	ƀ	05	ƀ	9B
12-11-8-4	0315	ƀ	05	ƀ	9C
12-11-8-5	0316	ƀ	05	ƀ	9D
12-11-8-6	0317	ƀ	05	ƀ	9E
12-11-8-7	0320	ƀ	05	ƀ	9F
11-0-8-1	0321	ƀ	05	ƀ	A0
11-0-8-2	0322	ƀ	05	ƀ	AA
11-0-8-3	0323	ƀ	05	ƀ	AB
11-0-8-4	0324	ƀ	05	ƀ	AC
11-0-8-5	0325	ƀ	05	ƀ	AD
11-0-8-6	0326	ƀ	05	ƀ	AE
11-0-8-7	0327	ƀ	05	ƀ	AF
12-11-0-8-1	0330	ƀ	05	ƀ	B0
12-11-0-1	0331	ƀ	05	ƀ	B1
12-11-0-2	0332	ƀ	05	ƀ	B2
12-11-0-3	0333	ƀ	05	ƀ	B3
12-11-0-4	0334	ƀ	05	ƀ	B4
12-11-0-5	0335	ƀ	05	ƀ	B5
12-11-0-6	0336	ƀ	05	ƀ	B6
12-11-0-7	0337	ƀ	05	ƀ	B7
12-11-0-8	0340	ƀ	05	ƀ	B8
12-11-0-9	0341	ƀ	05	ƀ	B9
12-11-0-8-2	0342	ƀ	05	ƀ	BA
12-11-0-8-3	0343	ƀ	05	ƀ	BB
12-11-0-8-4	0344	ƀ	05	ƀ	BC
12-11-0-8-5	0345	ƀ	05	ƀ	BD
12-11-0-8-6	0346	ƀ	05	ƀ	BE
12-11-0-8-7	0347	ƀ	05	ƀ	BF
12-0-9-8-2	0350	ƀ	05	ƀ	CA
12-0-9-8-3	0351	ƀ	05	ƀ	CB

Table D-7. EBCDIC/ASCII 80-Column Card Code Conversion (continued)

80-Column Card Code	ASCII Octal from ASCII Cards	ASCII Symbol from ASCII Cards	Fieldata Octal from ASCII Cards	Fieldata Symbol from ASCII Cards	EBCDIC Hexidecimal
12-0-9-8-4	0352	b	05	b	CC
12-0-9-8-5	0353	b	05	b	CD
12-0-9-8-6	0354	b	05	b	CE
12-0-9-8-7	0355	b	05	b	CF
12-11-9-8-2	0356	b	05	b	DA
12-11-9-8-3	0357	b	05	b	DB
12-11-9-8-4	0360	b	05	b	DC
12-11-9-8-5	0361	b	05	b	DD
12-11-9-8-6	0362	b	05	b	DE
12-11-9-8-7	0363	b	05	b	DF
11-0-9-8-2	0364	b	05	b	EA
11-0-9-8-3	0365	b	05	b	EB
11-0-9-8-4	0366	b	05	b	EC
11-0-9-8-5	0367	b	05	b	ED
11-0-9-8-6	0370	b	05	b	EE
11-0-9-8-7	0371	b	05	b	EF
12-11-0-9-8-2	0372	b	05	b	FA
12-11-0-9-8-3	0373	b	05	b	FB
12-11-0-9-8-4	0374	b	05	b	FC
12-11-0-9-8-5	0375	b	05	b	FD
12-11-0-9-8-6	0376	b	05	b	FE
12-11-0-9-8-7	0377	b	05	b	FF

The following table cross references 80-column ASCII cards codes to ASCII, Fieldata, and symbols from EBCDIC cards:

80-Column Card Code	ASCII Octal from EBCDIC Cards	ASCII Symbol from EBCDIC Cards	Fieldata Octal from EBCDIC Cards	Fieldata Symbol from EBCDIC Cards
12-8-7	041	!	02	]
11-8-2	041	!	055	!
8-1	0134	\	057	\
12-0	0173	{	054	?
11-0	0175	}	055	!
12-8-1	0135	]	02	]
11-8-1	0133	[	01	[
0-8-1	05		077	≠

Appendix E. Equipment Codes

Specific Mnemonic	Equipment Type
VIIIC	UNISERVO VIIIC Magnetic Tape (7-track)
VIC	UNISERVO VIC Magnetic Tape (7-track)
VIIICB	UNISERVO VIIIC Magnetic Tape (hardware translate 7-track)
VICB	UNISERVO VIC Magnetic Tape (hardware translate 7-track)
VIIIC9	UNISERVO VIIIC Magnetic Tape (9-track)
VIC9	UNISERVO VIC Magnetic Tape (9-track)
U12	UNISERVO 12 Magnetic Tape (7-track)
U14	UNISERVO 14 Magnetic Tape (7-track)
U16	UNISERVO 16 Magnetic Tape (7-track)
U30	UNISERVO 30 Magnetic Tape (7-track)
U12N	UNISERVO 12 Magnetic Tape (9-track)
U14N	UNISERVO 14 Magnetic Tape (9-track)
U12D	UNISERVO 12 Magnetic Tape (dual density 9-track)
U14D	UNISERVO 14 Magnetic Tape (dual density 9-track)
U16N	UNISERVO 16 Magnetic Tape (9-track)
U16D	UNISERVO 16 Magnetic Tape (dual density 9-track)
U20N	UNISERVO 20 Magnetic Tape (9-track)
U30D	UNISERVO 30 Magnetic Tape (dual density 9-track)
U32N	UNISERVO 32 Magnetic Tape (9-track)

Specific Mnemonic	Equipment Type
U34N	UNISERVO 34 Magnetic Tape (9-track)
U36N	UNISERVO 36 Magnetic Tape (9-track)
FH432	FH-432 Magnetic Drum
FH1782	FH-1782 Magnetic Drum
FII	FASTRAND II Magnetic Drum Subsystem
FIII	FASTRAND III Magnetic Drum Subsystem
F8460	UNIVAC 8460 Disk
BTBEQP	Back-to-Back Channel
D8414	UNIVAC 8414 Disk
D8424	UNIVAC 8424 Disk
D8425	UNIVAC 8425 Disk
D8430	UNIVAC 8430 Disk
D8433	UNIVAC 8433 Disk
D84050	UNIVAC 8405-00 Disk
D84054	UNIVAC 8405-04 Disk
D8440	UNIVAC 8440 Disk
D8450	UNIVAC 8450 Disk (without fixed head feature)
D8450F	UNIVAC 8450 Disk (with fixed head feature)
ARBDEV	Arbitrary Device
SYMDEV	Symbiont Devices
CTMCEQ	Communications Terminal Module Controller
CSP	Communications Symbiont Processor
GCS	Generalized Communication Subsystems
CNSOLE	Console
SPUEQP	System Partition Unit
ACUEQP	Access Control Unit

Appendix F. Run Setup Examples

The following examples illustrate runstreams used in executing various jobs.

F.1. ASSEMBLER EXAMPLE

The following runstream processes an assembly language program by calling the Assembler to assemble the source program, using the Collector to generate an absolute element, and then executing it.

```
1.  @RUN            GYP,665000/XYZ,QUAL,10,100
2.  @ASG,T          ATMOS
3.  @MASM,SD        ATMOS.TEN,ATMOS.TEN
4.                  AXR$
5.  PF              FORM      12,6,18
6.  S(0),BUF        RES       22
7.  MSG             'END OF CARD READER**'
8.  S(1),START
9.                  LA        A0,(EOF,BUF)
10.                 ER        READ$
11.                 LA        A0,(PF 1,BUF)
12.                 ER        PRINT$
13.                 J         START
14. EOF             LA        A0,(PF 1,4,MSG)
15.                 ER        PRINT$
16.                 ER        EXIT$
17.                 END       START
18. @MAP,IAL        ,ATMOS.TEN
19. IN              ATMOS.TEN
20. @XQT            ATMOS.TEN
21.       THIS PROGRAM READS
22.           A  VARIABLE
23.         NUMBER OF CARDS
24.                AND
25.           PRINTS  THEM.
26. @EOF
27. @PMD,E
28. @PASSWD         JJY
29. @FIN
```

Sample Assembly Language

Data to be Read When Program is Executed

1. @RUN control statement: GYP is the run-id; 665000 is the account number; XYZ is the user-id; QUAL is the project-id; estimated run time is 10 minutes; and estimated page count is 100.

2. Assign file ATMOS as a temporary file.

3. Assemble source language program TEN and produce a double-spaced listing consisting of both source language input and octal output. Insert symbolic element TEN into file ATMOS and produce a relocatable element called TEN in file ATMOS.

4. Items 4 through 17 are the source language input for program TEN.

18. Collect and produce absolute element TEN in file ATMOS. Accept the results of processing, even if errors are detected, and produce a comprehensive listing.

19. Input to the collector directing that element TEN in file ATMOS be included in the collection.

20. Execute absolute element ATMOS.TEN.

21. Items 21 through 25 are the data.

26. End-of-file mark used for indicating end of data input.

27. If the program terminates in error, a postmortem dump is taken.

28. User password JJY.

29. @FIN control statement terminates the run.

F.2. LEGENDRE POLYNOMIAL GENERATION

The following runstream executes the absolute element ABS produced by the collector from three relocatable elements produced in the run by the FORTRAN compiler and one element from a user-specified file.

1. @RUN controls statement: CLR is the run-id; 445 is the account number; XYZ is the user-id; 600 is the project-id; and nine minutes is the estimated run time.

2. Assign file FILEA in sector-format on any sector-formatted mass storage with an initial granule reserve of 10 tracks. FILEA is to be cataloged if the file is deassigned with a @FREE control statement or if the run terminates normally.

3. Assign previously cataloged file FILEB.

4. Call FORTRAN compiler. A new symbolic element EL1 is to be inserted into file FILEB. A listing is produced on the new source language and a relocatable output element EL2 is placed in FILEB.

5. Items 5 through 12 are the source language input deck.

13. Call FORTRAN compiler. A new source language element EL3 is inserted in FILEA. A comprehensive listing is produced of element EL3. The output relocatable element EL4 is placed in FILEA.

14. Items 14 through 24 are the source language input deck.

```
 1.  @RUN            CLR,445/XYZ,600,9
 2.  @ASG,C          FILEA.,F/10
 3.  @ASG,A          FILEB.
 4.  @FTN,IS         FILEB.EL1,.EL2
 5.  10      FORMAT(1H ,'END OF PROGRAM')
 6.  100     FORMAT( 2F10.3 )
 7.  200     READ(5,100,END=300) X,N
 8.          P = LEGED(X,N)
 9.          WRITE(6,20)P,X
10.  -       GO TO 200
11.  300     WRITE(6,10)
12.          END
13.  @FTN,IL        FILEA.EL3,FILEA.EL4
14.          REAL FUNCTION LEGED( Z, NU )
15.  10      FORMAT(1H ,'GAMMA FUNCTION ARGUMENT TOO LARGE')
16.  20      FORMAT(1H ,'GAMMA ARGUMENT ZERO OR NEGATIVE'  )
17.          REAL  NU
18.          LEGED = GAMMA(Z,NU,$110,$120)
19.          RETURN
20.  110     PRINT 10
21.          RETURN
22.  120     PRINT 20
23.          RETURN
24.          END
25.  @MASM,IL       FILEA.EL5,.EL6
          •
          •
          •
     SOURCE LANGUAGE INPUT
          •
          •
          •
26.  @MAP,IL        .SYM,.ABS
27.  IN             FILEB.EL2
28.  IN             FILEA.EL4
29.  IN             FILEA.EL6
30.  @XQT            ,.ABS
31.      0.100
32.      0.200
33.  @EOF
34.  @PASSWD        DWK
35.  @FIN
```

FORTRAN Source Language

25. Calls Assembler. A new source language element EL5 is inserted in FILEA, and a complete listing of the element is produced. A new relocatable element EL6 is placed in FILEA.

26. Call the Collector. Insert new MAP source language element SYM in the temporary program file (TPFS). Lines 11 through 15 are the source language statements which comprise element SYM. Produce the absolute element ABS in file TPF$ based on the supplied directives.

27. Include relocatable element EL2 from file FILEB in the collection to produce element ABS.

28. Include relocatable element EL4 from file FILEA in the collection to produce element ABS.

29. Include relocatable element EL6 from file FILEA in the collection to produce element ABS.

30. Execute absolute element ABS located in file TPF$.

31. Items 31 and 32 are data necessary for program execution.

33. End-of-file mark used for indicating end of data input.

34. User password DWK.

35. @FIN control statement terminates the run.

F.3. ASSEMBLER PROGRAM UPDATE EXAMPLE

In the following runstream, a program file is updated by using the Assembler and a new relocatable element generated. An absolute program is then generated. A separate file is assigned to accept results of program execution. The program is executed; then the separate file is deassigned and queued for output on a printer system.

```
1.  @RUN          AL5,888/XYZ
2.  @ASG,A        PROGS
3.  @ASG,CP       SPEC.,F2
4.  @MASM,S       PROGS.MURK(15),.PROGS,.MURK/ABER
5.  -23,25
6.                DL        A0,BUFFER+4          } Correction
7.                DSL       A0, 30               } statements to
8.                AND        A1,(077777777)      } source elements.
9.  -32,36
10. @MAP,I
11. IN            PROGS.
12. @BRKPT        PRINT$/SPEC
13. @XQT
14. 00000              3989          0010        }
15. 30103              3989          0020        }
16. 47712              3989          0030        } Data statements to
17. 60206              3988          0040        } be read at program
18. 69897              3986          0050        } execution.
19. 77815              3984          0060        }
20. 84510              3982          0070        }
21. @EOF
22. @BRKPT        PRINT$
23. @FREE         SPEC
24. @SYM          SPEC,,PR
25. @PASSWD       GMC
26. @FIN
```

1. @RUN control statement: AL5 is the run-id, 888 is the account number and XYZ is the user-id.

2. Assign previously cataloged file PROGS.

3. Assign file SPEC on sector-formatted mass storage and catalog it as a public file if one of the following occurs:

   a. the file is deassigned during the run by a @FREE control statement; or

   b. the run terminates without error.

4. Call the Assembler. Assembles element MURK(15) from file PROGS, and produces relocatable element PROGS and a new symbolic element MURK/ABER in file PROGS.

5. Replace lines 23 through 25 of the source language element MURK(15) with the lines which follow (items 6, 7, and 8) to produce MURK/ABER.

6. Lines 6 through 8 are correction statements to the source language element.

9. Delete lines 32 through 36 of the source language.

10. Calls the Collector.

11. Include file PROGS in the collection.

12. Close out the currently active print file and start a new print file SPEC.

13. Execute the program.

14. Lines 14 through 20 are data statements necessary for program execution.

21. End-of-file mark to terminate data input.

22. Print file SPEC is closed and a new system-defined print file is opened.

23. File SPEC is cataloged and made available for assignments by other runs.

24. File SPEC is queued for output on a printer subsystem.

25. User password GMC.

26. @FIN control statement terminates run.

F.4. FORTRAN PROGRAM EXAMPLE

The following example illustrates two runstreams. The first generates a program by use of the FORTRAN compiler, copies the symbolic element to tape, and then executes the program. The second references this tape file, uses the FORTRAN compiler to insert corrections and generate a new relocatable element, produces a new tape file, and then executes the program.

1. @RUN control statement: INIT is the run-id; 12345 is the account number; XYZ is the user-id and TR is the project-id.

2. Temporarily assign file TAPE to any magnetic tape unit; reel number is 12.

3. Call the FORTRAN compiler. A new source language input element PROG is inserted from the control stream into file TPFS and a listing is produced. The relocatable element produced, PROG, is placed in TPFS.

```
 1.  @RUN            INIT,12345/XYZ,TR
 2.  @ASG,T          TAPE,T,12
 3.  @FTN,IS         PROG,PROG
                •
 4.      FORTRAN SOURCE
                •
                •
 5.  @COPOUT,S       .PROG,TAPE.
 6.  @MARK           TAPE.
6A.  @FREE           TAPE.
6B.  @PASSWD         RCM
 7.  @XQT
                •
                •
 8.      DATA INPUT
                •
                •
                •
 9.  @PMD,E
10.  @FIN
11.  @RUN,/S         NEW,12345/XYZ,TR
12.  @ASG,T          TAPE.,T,12
13.  @ASG,T          PROGFILE.,F/2/POS/5
14.  @COPIN          TAPE.,PROGFILE.
15.  @FREE           TAPE.
16.  @PASSWD         RCM
17.  @FTN,S          PROGFILE.PROG,PROGFILE.RB,.PROG/A
                •
                •
                •
18.      CORRECTION STATEMENTS
                •
                •
19.  @ASG,T          NTAPE.,8C9,13
20.  @COPOUT         PROGFILE.,NTAPE.
21.  @FREE           NTAPE.
22.  @MAP,IL         .MAPSYM,.ABS
23.  IN              PROGFILE.RB
24.  @XQT            .ABS
25.             •
                •
                •
        DATA INPUT
                •
                •
                •
26.  @PMD,E
27.  @FIN
```

4. Source language statements.

5. Copy symbolic program element PROG from mass storage to tape file TAPE.

6. Mark an end-of-file (EOF) on the tape.

6A. Release the tape unit assigned as file TAPE. It is good practice to release a facility when it is no longer needed by the run.

6B. User password RCM.

7. Execute the program.

8. Data input necessary for program execution.

9. If the program terminates in error, a postmortem dump is taken.

10. @FIN control statement terminates the run.

11. @RUN control statement: The run is not to be opened until the previous run terminates: NEW is the run-id; 12345 is the account number; XYZ is the user-id and TR is the project-id.

12. Temporarily assign file TAPE to any tape unit; reel number is 12.

13. Assign file PROGFILE as a temporary file on mass storage with position granularity with an initial reserve of two granules and a maximum of five granules.

14. Copy input file TAPE from tape to mass storage file PROGFILE.

15. Release the tape unit assigned as file TAPE. It is good practice to release a facility when it is no longer needed by the run.

16. User password RCM.

17. Call the FORTRAN compiler. Compile element PROG from file PROGFILE; produce relocatable binary element RB and a new symbolic element PROG/A.

18. Correction statements to apply against PROG to produce PROG/A.

19. Temporarily assign file NTAPE to a UNISERVO VIIIC nine track tape unit; reel 13.

20. Copy program file PROGFILE on mass storage to element file NTAPE on tape.

21. Release the tape unit assigned as file NTAPE.

22. Calls the collector. Produce symbolic element MAPSYM which contains the MAP directives and produce absolute output element ABS according to the directives.

23. Include element RB from file PROGFILE in the collection.

24. Execute absolute program element ABS located in TPFS.

25. Data input necessary for the program.

26. Produce a postmortem dump if the program terminates in error.

27. @FIN control statement terminates the run.

F.5. TAPE FILE MANIPULATION

In the following runstream TPF$ is enlarged, then the tape file IN is copied into TPF$, and the Assembler is used to insert corrections into an element of file PARTIAL as well as to generate a new relocatable element. Two elements are deleted from file PARTIAL, a copy of it is made on tape file OUT, and then the elements of TPF$ are copied onto tape to form the first file. Then the contents of tape file OUT are checked.

```
1.  @RUN            BAL,09100/XYZ,TST,10,100
2.  @ASG,T          IN.,T,34
3.  @ASG,A          PARTIAL
4.  @FREE           TPF$.
5.  @ASG,T          TPF$.,F/2//800
6.  @COPIN          IN.
7.  @FREE           IN.
8.  @MASM,SO        PARTIAL.PART1,.PART2
9.  -34,50
              •
              •
              •
10.     CORRECTION STATEMENTS
              •
              •
              •
11.  @PASSWD        JJY
12.  @PRT,T         PARTIAL.
13.  @DELETE,R      .FIRST
14.  @DELETE,R      .SECOND
15.  @PACK
16.  @PREP
17.  @PRT,T
18.  @ASG,T         OUT.,T,BLANK
19.  @COPOUT,S      PARTIAL.,OUT
20.  @COPOUT        .,OUT.
21.  @FREE          PARTIAL.
22.  @REWIND        OUT.
23.  @ERS
24.  @COPIN         OUT
25.  @PRT,T
26.  @FIN
```

1. @RUN control statement: BAL is the run-id; 09100 is the account number; XYZ is the user-id; TST is the project-id; the estimated run time is 10 minutes; and estimated page count is 100.

2. Temporarily assign file IN to an available tape device; reel number is 34.

3. Assign file PARTIAL which was previously cataloged.

4. Release the facilities assigned to TPF$.

5. Temporarily assign TPF$ to a mass storage device with track granularity, an initial granule reserve of two tracks, and a maximum of 800 tracks.

6. Transfer program file IN from tape to mass storage file TPF$.

7. Release the facilities assigned to file IN.

8. Call the Assembler. Assemble element PART 1 from file PARTIAL and produce relocatable element PART 2. Produce a listing of only the octal output.

9. Delete lines 34 through 50 of the source language input element.

10. Correction statements.

11. User password JJY.

12. Call FURPUR processor. Produce an edited listing of the table of contents of file PARTIAL.

13. Call FURPUR processor. Mark relocatable element FIRST in TPF$ as deleted.

14. Call FURPUR processor. Mark relocated element SECOND in TPF$ as deleted.

15. Call FURPUR processor. Pack the nondeleted elements in TPF$.

16. Call FURPUR processor. Prepared an entry point table for TPF$.

17. Call FURPUR processor. Print the table of contents and the entry point table for TPF$.

18. Temporarily assign file OUT to an available tape device, request a blank reel.

19. Call FURPUR processor. Copy the symbolic elements from file PARTIAL into file OUT.

20. Call FURPUR processor. Copy the contents of TPF$ onto file OUT.

21. Release the facilities assigned to file PARTIAL.

22. Call FURPUR processor. Rewind file OUT to its load point.

23. Call FURPUR processor. Reset the next write location of TPF$ back to the start of the file and release its granules to the system.

24. Call FURPUR processor. Transfer file OUT from tape to mass storage file TPF$.

25. Call FURPUR processor. Produce an edited listing of the table of contents of file TPF$.

26. @FIN control statement terminates the run.

F.6. TAPE FILE BUILD EXAMPLE

The following runstream executes a program which builds a tape from an input tape file that contains relocatable binary elements. Then it expands TPF$ and uses it to list the table of contents of the second and third files of the output tape.

1. @RUN control statement with A priority; SYSCEN is the run-id; 400400 is the account number; XYZ is the user-id; BUILD is the project-id; and estimated run time is 15 minutes; estimated page count is 200.

2. Temporarily assign filename RB to any UNISERVO VIIIC, VIC or IVC tape unit; reel number is 3434C.

```
 1.  @RUN,A          SYSCEN,400400/XYZ,BUILD,15200
 2.  @ASG,T          RB.,C,3434C
 3.  @ASG,T          RO.,F2/2/POS/30
 4.  @MOVE           RB.,2
 5.  @COPIN          RB.,RO.
 6.  @FREE           RB.
 7.  @HDG            TOCS
 8.  @PASSWD         JJY
 9.  @PRT,T          RO.
10.  @USE            EXEC$,RO
11.  @HDG,P          EXEC-8    MAP
12.  @MAP,S          RO.EX8MAP,RO.EX8MAP
13.  @MAP,I          ,.SCH8PF
14.  IN              EXEC$.SCH8PF
15.  @XQT            SCH8PF
16.  @MSG            BUILD BOOTFILE
17.  @USE            B,BOOTFILE
18.  @ASG,TJ         BOOTFILE.,C,4545C
19.  @XQT            EXEC$.EX8MAP
20.  @ASG,TJ         OLDBOOT.,C,6565C
21.  @MOVE           OLDBOOT.,1
22.  @COPY,M         OLDBOOT.,B,5
23.  @FREE           OLDBOOT.
24.  @REWIND         B.
25.  @FREE           TPF$.
26.  @ASG,T          TPF$.,F2/2/POS/20
27.  @MOVE           B.,1
28.  @COPIN          B.
29.  @PRT,T
30.  @ERS
31.  @COPIN          B.
32.  @PRT,T
33.  @FIN
```

3. Temporarily assign file RO to sector-formatted mass storage unit with position granularity an initial reserve of two and a maximum size of 30 granules.

4. Call FURPUR processor. Move tape file RB over two EOF marks.

5. Transfer tape file RB to mass storage file RO.

6. Release the facilities assigned to file RB.

7. Print the heading TOCS at the top of each page.

8. User password JJY.

9. Call FURPUR processor. Produce an edited listing of the table of contents of file RO.

10. The internal filename EXEC$ is attached to file RO and the file can be referenced by either name throughout the remainder of this run.

11. Print the heading EXEC 8 MAP beginning with page one instead of the print file's current page number.

12. Call the Collector. Use the directives in symbolic element EX8MAP in file RO to produce an absolute element of the same name.

13. Call the Collector. Use the directive which immediately follows in the runstream and produce absolute element SCH8PF in file TPF$.

14. Relocatable element SCH8PF in file EXEC$ is to be included in the collection.

15. Execute absolute element SCH8PF which is in file TPF$.

16. Transmit the message BUILD BOOTFILE to the operator's console.

17. Execute element EX8MAP in file EXEC$.

18. Internal filename B is attached to file BOOTFILE and the file can be referenced by either name throughout the remainder of the run.

19. Temporarily assign file BOOTFILE to any UNISERVO VIIIC, VIC, or IVC tape unit; reel number is 4545C and must not be labeled tape.

20. Temporarily assign file OLDBOOT to any UNISERVO VIIIC, VIC, or IVC tape unit; reel number is 6565C and must not be labeled tape.

21. Call FURPUR processor. Move tape file OLDBOOT over one EOF mark.

22. Copy five files from OLDBOOT to tape file B and place one EOF mark after each file copied to tape file B.

23. Release the facilities assigned to file OLDBOOT.

24. Rewind tape file B to its load point.

25. Release the facilities assigned to TPF$.

26. Temporarily assign TPF$ to sector-formatted mass storage with initial reserve of two granules, position granularity, and a maximum size of 20 granules.

27. Move tape file B forward past one EOF mark.

28. Transfer tape file B to mass storage file TPF$.

29. List the table of contents for TPF$.

30. Reset the next write location of TPF$ back to the start of the file.

31. Transfer tape file B to mass storage file TPF$.

32. Print the table of contents of file TPF$.

33. @FIN control statement terminates the run.

F.7. UNISCOPE TERMINAL BREAKPOINT TO HIGH SPEED PRINTER

In the following example a UNISCOPE display terminal is activated, a PAGEWRITER printer assigned to it, and then a runstream submitted. During the run a file MAX is assigned which is to be cataloged. Then two source elements are generated; one is an Assembler language program and the other is a partial runstream which processes the first element and also has the control statements necessary to output to the high speed printer. The second element, HELPER, is then added to the runstream. The terminal operator then gets output from the high speed printer, modifies the original source element, and processes it by adding HELPER to the runstream again. The operator receives output from the high speed printer, terminates the run and then deactivates the terminal.

1. Terminal identifier: Activates terminal and allows for station communication with the central site.

1A. Enter user-id/password.

2. PAGEWRITER printer identifier: Assigns the PAGEWRITER printer with this identifier to this terminal.

3. @RUN control statement: PLANCK is the run-id; 662619 is the account number; RADIATION is the project-id; the estimated run time is 5 minutes; and estimated page count is 200.

4. Message is sent to the central site operator informing the user of the account number. The W option requires that the central site operator acknowledge the message.

5. Assign a file MAX on disk storage to be cataloged when a @FREE statement is encountered or when the run terminates normally. One granule is to be reserved initially, the granule size is that of a track, a maximum of 50 granules is allowed before an error termination occurs, and the pack-id of the disk is 775.

6. The Text Editor is called. The I option indicates that it is initially in the input mode. The input is used to generate a source element RADIATION in the file MAX.

7. Items 7 through 51 are the source program statements. Item 7 calls a PROC which defines the registers and j-designators in terms of mnemonic.

16. The PROC E$PKTF is used to define a packet which is used by the EDIT$ routines for composing the contents of the output buffer OUTBUF.

22. This is a call on the FORTRAN library function NEXPB$ which raises a double-precision base to a double-precision power.

27. This is a call on the FORTRAN library function DEXP which is the double-precision exponential function.

32. This is a call on the FORTRAN library function which raises a double-precision base to an integer power.

38. This item initiates the editing mode of the EDIT$ routine using packet EDPKT.

39. This item sets the column pointer to column 10, leaving columns 1 through 9 blank filled.

40. This item converts the double-precision contents of address XOO into Fieldata and insert it into the output buffer.

```
 1.  SSU021
1A.  ABCDEF/123456
 2.  @@PRNT 1
 3.  @RUN            PLANCK,662619,RADIATION,5200
 4.  @MSG,W          MY ACCOUNT NUMBER IS 662619
 5.  @ASG,C          MAX,F14/1//50,775
 6.  @ED,I           MAX.RADIATION
 7.                  AXR$
 8.  P               FORM      12,6,18
 9.  NUM             EQU       700
10.  S(0),X00        +0.050D
11.  XINT            +0,005D
12.  MINUS1          -1.0D
13.  MINUS5          -5.0D
14.  TEMP            RES       2
15.  OUTBUF          RES       22
16.  EDPKT           E$PKTF    22,OUTBUF
17.  MSG             'VALUES OF THE PLANCK RADIATION FUNCTION'
18.  S(1),START
19.                  LA        A0,(P 077,7,MSG)
20.                  ER        PRINT$
21.                  LR,U      R5NUM-1
22.  LOOP            LMJ       X11,NEXPB$
23.                  +         X00
24.                  +         MINUS1
25.                  +         0,0
26.                  DS        A0,TEMP
27.                  LMJ       X11,DEXP
28.                  +         TEMP
29.                  +         0,0
30.                  DFA       A0,MINUS1
31.                  DS        A0,TEMP
32.                  LMJ       X11,NEXP9$
33.                  +         X00
34.                  MINUS5
35.                  +         0,0
36.                  DFD       A0,TEMP
37.                  DS        A0,TEMP
38.                  E$DIT     EDPKT
39.                  E$COL     10
40.                  E$FLF2    10*/6+4,X00
41.                  E$SKIP    10
42.                  E$FLF2    15*/6+8,TEMP
43.                  E$DITX
44.                  LA        A0,(P 1,22, OUTBUF)
45.                  ER        PRINT$
46.                  DL        A0,X00
47.                  DFA       A0,XINT
48.                  DS        A0,X00
49.                  JGD       R5LOOP
50.                  ER        EXIT
51.                  END        START
```

Source Element

```
52.  @ED,I           MAX.HELPER
53.
54.  I @ASG,CP       PRINTER,F2///2000
55.  I @BRKPT        PRINT$/PRINTER
56.  I @MASM,L       MAX.RADIATION,.RB
57.  I @MAP,IL       ,MAX.CONSTANT
58.  I    IN         MAX.RB
59.  I @XQT          MAX.CONSTANT
60.  I @BRKPT        PRINT$
61.  I @FREE         PRINTER
62.  I @SYM,U        PRINTER,,PR
63.  I @SYM          PRINTER,,PR
64.  EXIT
65.  @ADD,L          MAX.HELPER
66.  @ED,U           MAX.RADIATION
67.  4
68.  C/0.05/0.10/
69.  +1
70.  C/5/1/
71.  EXIT
72.  @ADD,L          MAX.HELPER
73.  @FIN
74.  @@TERM
```

41. This item moves the column pointer forward another 10 places, thus leaving them blank filled.

42. The double-precision value stored in location TEMP in converted into a Fieldata representation and inserted into the output buffer.

43. This statement terminates the EDIT$ editing mode.

44. This statement and the following one outputs the contents of the output buffer OUTBUF.

52. Terminate input to the Text Editor and calls the Text Editor. The I option indicates that the source input is to be used to build the element HELPER in file MAX.

53. An input of zero length (blank line) changes from input mode to edit mode.

54. Items 54 through 63 are control statements to be entered into element HELPER. The I followed by a space indicates that which follows is to be input into the source element. This allows statements starting with a master space in column one to be used as input – if they were input while the Text Editor was in the input mode with the master space in column one, then the Text Editor would have been terminated. Later this element is @ADDed to the runstream. When added (@ADD), item 54 catalogs PRINTER as a public file on sector-format.

55. This item closes out the current print file and places future print output into file PRINTER.

56. This item calls the Assembler using element RADIATION of file MAX for its source input and then places the generated relocatable binary in element RB of file MAX. The L option means that a comprehensive listing is requested.

57. This item calls the Collector; the absolute element is to be named CONSTANT and placed in file MAX; the I option indicates that source statements follow which direct the collection and the L option requests the most comprehensive listing available.

58. This item directs the Collector to use the element RB in file MAX.

59. This item requests the execution of absolute element CONSTANT of file MAX.

60. This item terminates print output going into file PRINTER. Future output goes directly to the terminal.

61. This item catalogs the file PRINTER and then deassigns it from the run.

62. This item enters the file PRINTER in the high speed printer symbiont queue. This U option indicates that the file is not to be decataloged.

63. This item enters file PRINTER in the high speed printer symbiont queue again, so that two copies are made. The absence of the U option means that the file is to be decataloged after being printed. Items 62 and 63 can be replaced by @SYM PRINTER,2,PR.

64. This item terminates the Text Editor. The element HELPER is now entered in file MAX containing statements 54 through 63.

65. The control statements stored in element HELPER are now added into the runstream. The L option means that as each control statement is encountered, it is listed at the terminal. Normally the terminal user would go out to the high speed printer and pick up two copies of the run output at this time. The user decides to modify the program at this point as shown below.

66. This item calls the Text Editor. The U option indicates that a modification is to be made to source element RADIATION of file MAX.

67. This item directs the Text Editor to direct its editing to line 4 of the source element which it then prints at the terminal.

68. This item directs the Text Editor to search for a character string of "0.05" in the fourth line by searching from left to right. When found, it is to be removed and replaced with the string "0.10".

69. This item directs the Text Editor's attention to the next line of source code.

70. This item directs a search of line 5 for the first character 5. When found, it is to be replaced by a "1".

71. This item terminates the Text Editor. At this time the changes previously requested are made to the source element by generating a new higher cycle number.

72. As with item 65 the control statements in element HELPER are now added to the runstream. However, the processing is done on the updated element RADIATION with the resultant different output at program execution.

73. This item terminates the run.

74 This item inactivates the UNISCOPE terminal and releases the PAGEWRITER printer from the terminal.

F.8. START RUN EXAMPLE

In the following example a DCT 500 remote terminal is activated and then a run is initiated. A file RUNFILE is cataloged and two source elements are generated in it. One is a FORTRAN program and the other is a remote batch run which is started later in the run. After the stored run is completed the FORTRAN program is modified and the run started again.

```
 1.  UD108Z
1A.  ABCDEF/123456
 2.  @RUN            MINE,41972,SPECTRO,1,20
 3.  @MSG            MY ACCOUNT NUMBER IS 41972
 4.  @ASG,CP         RUNFILE,F2//TRK/100
 5.  @ED,I           RUNFILE.PROGRAM
 6.  100     FORMAT(1H1,40X,22HTABLE OF (WXSIN(W))**2)
 7.  120     FORMAT(1H ,4(F8.6,4X,F12.8,14X) )
 8.          PRINT 100
 9.          TWOPI = 6.2831853
10.          DO 500 I = 1, 2501
11.          F1 = (I - 1)*0.0001
12.          F2 = F1 + 0.25
13.          F3 = F1 + 0.50
14.          F4 = F1 + 0.75
15.          C1 = (TWOPI * F1 * SIN(TWOPI*F1))**2
16.          C2 = (TWOPI * F2 * SIN(TWOPI*F2))**2
17.          C3 = (TWOPI * F3 * SIN(TWOPI*F3))**2
18.          C4 = (TWOPI * F3 * SIN(TWOPI*F3))**2
19.  500     PRINT 120,F1,C1,F2,C2,F3,C3,F4,C4
20.          END
21.  @ED,I           RUNFILE.RUNSTREAM
22.
23.  I @RUN,/TP      FAST,007,SPECTRO,5100
24.  I @MSG          MY ACCOUNT NUMBER IS 007
25.  I @HDG,P           ***LINE WIDTH FUNCTION VALUES***
26.  I @FTN,S        RUNFILE.PROGRAM,TPF$.RB
27.  I @XQT
28.  EXIT
29.  @FREE           RUNFILE
30.  @START          RUNFILE.RUNSTREAM
31.  @ASG,AX         RUNFILE
32.  @ED,U           RUNFILE.PROGRAM
32A. +1
33.  CHANGE /SIN/COS/  15
34.  EXIT
35.  @FREE           RUNFILE
36.  @START          RUNFILE.RUNSTREAM
37.  @FIN
38.  @@TERM
```

FORTRAN source language

1. This item is an identifier which is transmitted from the DCT 500 as one step in activating the station.

1A. Enter user-id/password.

2. Run control statement: the run identifier is MINE, the account to which the costs of the run are to be charged is 41972, the project is SPECTRO, the expected run time is one minute, and the expected number of pages of output is 20.

3. This item transmits a message to the central site operator.

4. A file RUNFILE is assigned and is to be cataloged public if the run terminates normally or by a @FREE statement. It is to reside on sector-format, have no initial reserve, have track granularity, and use up to a maximum of 100 granules.

5. The Text Editor is called. The I option means that it is initially in the input mode and that a source element PROGRAM is to be built in file RUNFILE.

6. Items 6 through 20 are the source input statements which are to be stored in element PROGRAM.

21. This item terminates the previous Text Editor input (as any Executive command), the previously requested source element is built, and then the Text Editor is called again. The I option indicates that the input mode is desired and that source element RUNSTREAM is to be built in file RUNFILE.

22. This is a blank statement used for changing the Text Editor mode from input to the edit mode.

23. Items 23 through 27 are the FORTRAN source input statements which constitutes a stored runstream which later is started as a batch run. The I in column one indicates that that which follows is to be stored in the element. If the master space or at symbol (@) had been in column one the Text Editor would have been terminated and the statement interpreted as a command to be operated on immediately. The stored run statement has a run identifier of FAST, computer charges are charged to account 007, the project is SPECTRO, expected to run 5 minutes and a maximum of 100 pages of output is allowed. The T and P options indicate that the run is to be terminated if either the maximum time or page limits are exceeded.

24 This item sends the operator a message when the run is started.

25. This item initiates a page heading of "***LINE WIDTH FUNCTION VALUES***". The P option causes the heading to include the current date and page numbers.

26. This item calls the FORTRAN compiler which receives its source input from element PROGRAM of file RUNFILE and places the generated code in relocatable element RB of file TPF$. The S option indicates that an abbreviated listing is required.

27. This item terminates input to the FORTRAN compiler, calls the Collector which generates an absolute element in file TPF$ called NAME$ using the relocatable element RB, and then executes that absolute element.

28. This item terminates the Text Editor. At this time the element RUNSTREAM is built in file RUNFILE.

29. This item deassigns the file RUNFILE from the run and catalogs it.

30. This item starts the run stored in source element RUNSTREAM of file RUNFILE. The run is initiated as a batch run in the same manner as if the statements in the RUNSTREAM had been punched cards entered via the card reader.

33. This item indicates that the characters "SIN" are to be searched for from left to right and if found replaced by the characters "COS". This search is to be repeated so that in the first 15 lines the above replacement is made.

34. This terminates further input to the Text Editor and the requested changes are made by generating a new cycle of element PROGRAM.

35. This item deassigns the file RUNFILE from the run.

36. This item starts the run stored in source element RUNSTREAM as in item 30 above, except that the FORTRAN compiler processes an updated source element PROGRAM.

37. This item terminates the run.

38. This item deactivates the terminal and clears the telephone line.

Index

Term	Reference	Page
A		
Abbreviations	2.2.2	2-13
ABORT$	4.3.2.3	4-8
ABSAD$	4.7.4	4-27
Absolute device, assignment	3.7.1.3.2	3-56
Absolute element	2.6.4	2-25
ACSF$	4.10.1.2	4-76
Activity		
activation	4.3.3.4	4-11
changing priorities	10.4.2.1	10-7
changing to real-time	4.3.5.1	4-18
creating	4.3.1.1	4-6
creating with timed wait	4.3.1.2	4-7
deactivation	4.3.3.3	4-10
dedication	4.3.1.3	4-7
interrupt	6.1.2	6-5
interrupt, priority reduction	10.4.2.3	10-8
interrupt, inter-activity	4.3.3.5	4-11
joining	4.3.3.1	4-10
naming	4.3.3.2	4-10
program control	4.3	4-6
real-time	10.6.2.2	10-12
reducing interrupt priority	6.3.9	6-12
removing real-time status	4.3.5.2	4-19
synchronization	4.3.3	4-9
	4.11.7	4-90
termination		
error	4.3.2.4	4-9
normal	4.3.2.1	4-8
timed wait	4.3.6	4-19
user leveling	4.3.7	4-19
ACT$	4.3.3.4	4-11
release ESI activity control	9.5	9-32
synchronization	4.11.7.2	4-90
ADACT$	9.5	9-33
@ADD	3.10.1	3-70
	C.4.4	C-37
status code	C.4.3	C-36
Addressing		
file	7.2.3	7-3
inter-bank	4.11.3	4-88
ADED$	4.3.1.3	4-7
Alternate file, ASCII images	5.2.4	5-6

Term	Reference	Page
Alternate print file		
ASCII control functions	5.4.4	5-20
ASCII images	5.3.4	5-10
Fieldata control functions	5.4.3	5-19
Fieldata images	5.3.3	5-9
Alternate punch file		
ASCII control functions	5.4.8	5-21
ASCII images	5.3.8	5-13
Fieldata control functions	5.4.7	5-21
Fieldata images	5.3.7	5-12
Alternative file, Fieldata images	5.2.3	5-5
APCHCA$	5.4.8	5-21
APCHCN$	5.4.6	5-20
APNCHA$	5.3.8	5-13
APRINT$	5.3.2	5-9
APRNTA$	5.3.4	5-10
APRTCA$	5.4.4	5-20
APRTCN$	5.4.2	5-19
APUNCH$	5.3.6	5-12
Arbitrary device		
assignment	3.7.1.3	3-54
auxiliary storage interface	6.8.7	6-45
initiate and exit with interrupt (IOAXI$)	6.8.3	6-41
initiate, return control immmediately (IOARB$)	6.8.2	6-40
interface	6.8	6-36
I/O packet	6.8.1	6-37
I/O packet format	6.9.1	6-47
I/O path selection	6.8.6	6-44
MSA tape	6.8.4	6-41
packet format	Figure 6-3	6-37
tape considerations	6.8.5	6-43
AREADA$	5.2.4	5-6
AREAD$	5.2.2	5-4
ASCII		
special characters	C.3	C-4
to Fieldata conversion	Table D-2	D-3
ASCII control functions		
alternate print file	5.4.4	5-20
alternate punch file	5.4.4	5-20
print file	5.4.2	5-19
punch file	5.4.6	5-20
ASCII images		
alternate file	5.2.4	5-6
alternate print file	5.3.4	5-10
alternate punch file	5.3.8	5-13
dynamic request of control statements	4.10.1	4-74
printing	5.3.2	5-9
punching	5.3.6	5-12
reading	5.2.2	5-4
@ASG		
files and peripheral devices	3.7.1	3-37
magnetic tape options	3.7.1.2	3-47
magnetic tape	Table 3-5	3-46
sector-formatted mass storage	Table 3-4	3-38
sector-formatted files	3.7.1.1	3-39
status codes	Table C-1	C-16
word-addressable mass storage, normal	3.7.1.3	3-54
Assembler		
example	F.1	F-1
update example	F.3	F-4
ATREAD$	5.2.6	5-8
Auxiliary storage interface via ADI	6.8.7	6-45
AWAIT$	4.3.3.1	4-10
synchronization	4.11.7.2	4-90

Term	Reference	Page
B		
Bank		
active	3.4.4.4.5	3-24
additional space	4.11.1.2	4-87
address limits	4.11.4	4-89
common access	3.4.4.2.4	3-20
common usage	4.11.1.1	4-85
dynamic usage	4.11.1	4-85
initial load	3.4.4.4.4	3-24
initially based common	3.4.4.2.3	3-20
inter-addressing	4.11.3	4-88
collection	4.11.3.1	4-88
Collector produced tables	4.11.3.3	4-89
register basing	4.11.3.2	4-89
lowest address	3.4.4.2.2	3-19
referencing	3.4.4.4	3-21
active	3.4.4.4.5	3-24
initial load	3.4.4.4.4	3-24
static vs. dynamic	3.4.4.4.3	3-23
switching between	3.4.4.4.2	3-22
visible	3.4.4.4.1	3-21
unbasing	3.4.4.4.7	3-24
BANK$, Bank Descriptor Index (BDI) retrieval	4.8.4	4-40
Batch processing	1.3.1.1	1-3
BDSPT$	4.7.6	4-29
Binary		
hexidecimal conversion	D.5	D-7
time and date	4.5.2	4-21
Branching, from within a runstream	3.10.4.3	3-76
Breakpoint, setting user	4.10.4	4-80
@BRKPT		
alternate symbiont file	3.6.2.2	3-30
example of usage	3.6.4	3-33
primary output file	3.6.2.1	3-29
status code	Table C-3	C-35
Buffer		
processing	9.7.2	9-35
real-time operations	10.3	10-2
real-time size	10.3.4	10-3
single mode for I/O	9.4.1.6	9-18
C		
CADD$	9.4.2.3	9-29
Card reader		
standard mode control	3.6.5	3-34
9000 mode control	3.6.6	3-36
@CAT, status code	C.2	C-16
Cataloged file		
recovery	12.8	12-25
@CAT	3.7.3	3-58
options	Table 3-6	3-57
status code	Table C-1	C-16
@@CDI	A.2	A-8
DCT 1000	8.2.2.2	8-16
	8.2.2.3	8-19
@@CDO	A.2	A-8
DCT 1000	8.2.2.2	8-16
	8.2.2.3	8-19
CEND$	4.9.4.2	4-63
CGET$	9.4.2.2	9-29
Checkpoint		
control statement	11.2.1	11-2
error codes	Table 11-3	11-7
error messages	Table 11-4	11-9
Executive Request	11.2.2	11-2
introduction	11.1	11-1
@CKPT	11.2.1	11-2
Checkpoint/Restart	Section 11	
CJOIN$	9.4.2.4	9-30
@CKPT	11.2.1	11-2
CLIST$	5.5	5-22

Term	Reference	Page
Clocking	12.6	12-21
@@CM	8.1.1.3.1	8-4
CMD$	9.4.1.2	9-16
CMH$	9.4.1.9	9-21
CMI$	9.4.1.3	9-17
CMO$	9.4.1.4	9-18
CMSA$	9.4.1.5	9-18
CMS$	9.4.1.1	9-15
usage within common banks	4.11.5.3	4-90
CMT$	9.4.1.10	9-21
Coarse scheduler	12.5.4	12-13
@COB	3.9	3-67
@COL		
standard card reader	3.6.5	3-34
9000 card reader	3.6.6	3-36
Collector tables	4.11.3.3	4-89
Collector (MAP) processor	1.4.1	1-7
Common bank		
access	3.4.4.2.4	3-20
data protection within	4.11.7	4-90
dumping	4.11.6	4-90
Executive Requests within	4.11.5	4-89
reload	4.8.4.2	4-42
	4.11.1.1.1	4-86
usage	4.11.1.1	4-85
Common Data Bank (CDB) contingencies		
processing	4.9.6.2	4-69
queuing for user program	4.9.6.3	4-70
registration	4.9.6.1	4-69
termination		
abnormal	4.9.6.4	4-71
notification of TRMRG$	4.9.6.5	4-72
Communications		
completion activities	9.5	9-32
console	4.6	4-22
equipment	9.1.1	9-1
exiting from an ESI activity	9.5	9-32
handler support operations	9.4.1	9-15
idle line monitor	9.6.1	9-33
interrupt response	9.7.1	9-34
modes of operation	9.1.2	9-3
operator	1.3.3.5	1-5
pools	9.4.2	9-22
timing considerations	9.7	9-34
Communications Control Routine (CCR)		
debugging	8.7.1.5	8-42
ER RSI$	8.7.1	8-40
initialization	8.7.1.1	8-40
input function	8.7.1.2	8-41
output function	8.7.1.3	8-41
termination	8.7.1.4	8-42
Communications Handler		
assigning line terminal devices	9.2	9-4
completion activities	9.5	9-32
error codes for line terminal contingencies	9.9	9-37
idle line monitor	9.6.1	9-33
idle line polling	9.6.2	9-34
information analysis	9.8	9-36
introduction	9.1	9-1
modes of operation	9.1.2	9-3
operations	9.4	9-15
timing considerations	9.7	9-34
dialing (CMD$)	9.4.1.2	9-16
dual mode for input operations	9.4.1.8	9-21
hangup (CMH$)	9.4.1.9	9-21
initialization (CMS$)	9.4.1.1	9-15
input (CMI$)	9.4.1.3	9-17
output (CMO$)	9.4.1.4	9-18
pool mode for I/O operations	9.4.1.7	9-19
send and acknowledge (CMSA$)	9.4.1.5	9-18

Term	Reference	Page
single buffer mode for I/O operations	9.4.1.6	9-18
support operations	9.4.1	9-15
termination (CMT$)	9.4.1.10	9-21
Communications pools		
altering (ROUTE$)	9.4.3	9-31
establishing (CPOOL$)	9.4.2.1	9-26
expanding (CJOIN$)	9.4.2.4	9-30
releasing (CREL$)	9.4.2.5	9-30
removing buffers (CGET$)	9.4.2.2	9-29
returning buffers (CADD$)	9.4.2.3	9-29
Communications peripherals, FITEM$ request packet	7.2.6.5	7-11
COM$	4.6.1	4-22
Condition word		
control	3.10.4.1	3-74
	4.4	4-20
retrieval	4.4.2	4-20
setting	4.4.1	4-20
testing	3.10.4.2	3-75
Conditional statements	3.10.2	3-72
COND$	4.4.2	4-20
@@CONS	8.1.1.3.1	8-4
	A.2	A-8
Console		
communications	4.6	4-22
interrupt handling real-time	10.4.2.6	10-9
output	4.6.1	4-22
solicited input	4.6.1	4-22
unsolicited input	4.6.2	4-24
@@CONT	8.1.1.3.1	8-4
	A.2	A-8

Term	Reference	Page
Contingency		
abort	4.9.4.5	4-66
additional considerations	4.9.4.4	4-64
common data banks	4.9.6	4-68
definition	2.2.1	2-4
	4.9.1	4-54
error types	4.9.2.1	4-57
ESI	4.9.5	4-66
hardware fault	4.9.4.6	4-66
line terminal	9.9	9-37
mode termination	4.9.4.2	4-63
mode termination and return	4.9.4.3	4-63
multiple	4.9.4.4	4-64
nested	4.9.4.4	4-64
registration	4.9.3	4-59
types	Table 4-3	4-55
types and standard action	4.9.2	4-54
Control		
activity and program	4.3	4-6
queuing	6.1.3	6-5
Control characters		
DCT 1000	8.2.2	8-13
DCT 500	8.2.1	8-9
Teletypewriter	8.2.1	8-9
UTS 100/200	8.2.2	8-13
UTS 400	8.2.2.4	8-21
Control functions, ASCII		
alternate print file	5.4.4	5-20
alternate punch file	5.4.8	5-21
print file	5.4.2	5-19
punch file	5.4.6	5-20
Control functions, Fieldata		
alternate print file	5.4.3	5-19
alternate punch file	5.4.7	5-21
print functions	Table 5-2	5-15
punch file	5.4.5	5-20
punch functions	Table 5-4	5-21
Control register, parity error interrupts	12.7.3.1	12-23
Control statement		
annotation	3.2.4	3-4
continuation	3.2.5	3-5
data preparation	3.8	3-65

Term	Reference	Page
default rules	3.2.7	3-5
demand symbiont summary	A.2	A-8
dynamic request of	4.10.1	4-74
facility	3.7	3-37
format	3.2	3-1
interpreter (CSI)	12.5.3	12-12
labeling	3.10.3	3-72
language processor	3.9	3-67
listing user-defined	5.5	5-22
message	3.5	3-25
notation	2.3.2	2-16
processor	3.9	3-67
scheduling	3.4	3-9
summary	Table 3-1	3-5
	A.1	A-1
transparent	3.2.8	3-6
	8.1.1.3.2	8-6
	A.1	A-1
	A.2	A-8
@ADD	3.10.1	3-70
@ASG	3.7.1	3-37
@BRKPT	3.6.2	3-29
@CAT	3.7.3	3-58
@CKPT	11.2.1	11-2
@COB	3.9	3-68
@COL	3.6.5	3-34
@ENDCL	3.6.5	3-35
@ENDF	3.8.2	3-66
@EOF	3.4.4.3	3-20
@FILE	3.8.1	3-65
@FIN	3.4.2	3-15
@FREE	3.7.4	3-60
@HDG	3.6.1	3-28
@JUMP	3.10.4.3	3-76
@LOG	3.5.2	3-27
@MODE	3.7.2	3-57
@MSG	3.5.1	3-25
@QUAL	3.7.6	3-64
@RSTRT	11.3.1	11-2
@RUN	3.4.1	3-9
@SETC	3.10.4.1	3-74
@START	3.4.3	3-16
@SYM	3.6.3	3-31
@TEST	3.10.4.2	3-75
@USE	3.7.5	3-63
@XQT	3.4.4	3-18
Conventions		
calling sequence	4.1.2	4-1
control statement notation	2.3.2	2-16
notational	2.3.1	2-15
Conversational mode, Fieldata images	5.2.5	5-7
Conversational Time Sharing (CTS)	1.4.10	1-8
Conversion tables	Appendix D	
ASCII-to-Fieldata	Table D-2	D-3
binary/hexadecimal	D.5	D-7
Fieldata-to-ASCII	Table D-1	D-2
octal/decimal	Table D-5	D-8
80-column card code, Fieldata	D.7	D-12
80-column card code, ASCII	D.8	D-14
CPOOL$	9.4.2.1	9-26
usage within common banks	4.11.5.3	4-90
CQUE$	4.9.6.3	4-70
@@CQUE	8.1.1.3.1	8-4
	A.2	A-8
CREG$	4.9.3.2	4-61
CREL$	9.4.2.5	9-30
CRTN$	4.9.4.3	4-63
CSF$	4.10.1.1	4-74
real-time	10.5	10-10
status codes	C.4	C-35
CTMC	9.1.1.1	9-1
CULL processor	1.5.1	1-9
Cursor/SOE coordinates, UNISCOPE 100/200, UTS 400	D.4	D-6
Cycle, symbolic element	2.6.5	2-25
C$TS	4.3.4.4	4-15
C$TSA	4.3.4.6	4-16
C$TSQ	4.3.4.5	4-15
C/SP	9.1.1.2	9-2

Term	Reference	Page
D		
DACT$	4.3.3.3	4-10
synchronization	4.11.7.2	4-90
Data		
program separation	3.4.4.3	3-20
protection	4.11.7	4-90
DATA processor	1.4.4	1-8
DATE$	4.5.1	4-21
Day clock	12.6.2	12-22
@@DCT	8.2.1.5	8-12
	A.2	A-8
DCT 1000		
demand symbiont control statements	8.2.2.3	8-17
operational considerations	8.2.2.2	8-16
DCT 500/475		
demand symbiont	8.2.1	8-9
interrupting output	8.2.1.4	8-11
operational considerations	8.2.1.1	8-9
paper tape operations	8.2.1.2	8-10
special characters	8.2.1.3	8-11
Teletypewriter mode	8.2.1.6	8-12
@@DCT	8.2.1.5	8-12
Default rules	3.2.7	3-5
Definitions	2.2	2-1
Demand		
batch sharing	12.5.5.3	12-18
processing	1.3.1.2	1-3
	Section 8	
run example	8.6	8-34
terminal termination	8.1.1.4	8-9
Demand symbiont		
DCT 1000	8.2.2	8-13
DCT 500	8.2.1	8-9
DCT 500, Teletypewriter mode	8.2.1.6	8-12
general operation	8.1.1	8-2
Teletypewriter	8.2.1	8-9
UNISCOPE 100/200	8.2.2	8-13
UTS 400	8.2.2.4	8-21
Demand terminal		
demand symbiont interface statements	8.1.1.3	8-3
demand symbionts	8.2	8-9
general operational procedures	8.1.1	8-2
initialization	8.1.1.1	8-2
messages	8.1.1.3.3	8-6
modes of operation	8.1.1.2	8-3
termination	8.1.1.4	8-9
user techniques	8.5	8-32
Device handler, input/output	1.3.3.6	1-5
	Section 6	
Device Status Buffer (DSB)	6.9.2	6-53
Devices		
assigning line terminal	9.2	9-4
releasing peripheral	3.7.4	3-60
Diagnostic messages, runstream	C.1	C-1
Disk		
FITEM$ request packet	7.2.6.6	7-15
free format	6.8.4	6-41
functions	6.6	6-32
labeling	7.4	7-28
mass storage	6.6	6-32
removable recovery/registration	12.9	12-26
Dispatcher	12.5.6	12-19
Distributed Communications Processor (DCP)	9.1.3	9-3
DOC processor	1.5.2	1-9

Term	Reference	Page
Dump, main storage snapshot	4.10.3	4-79
Dynamic Allocator (DA)		
demand/batch sharing	12.5.5.3	12-18
dynamic main storage allocation	12.5.5.2	12-16
general overview	12.5.5.1	12-15
time sharing	12.5.5.4	12-18
Dynamic bank usage	4.11	4-85
Dynamic dump, options	3.4.1.1	3-13
E		
EABT$	4.3.2.4	4-9
ED processor	1.4.7	1-8
EDJS$	4.8.6	4-48
Element		
names	2.6.4	2-25
reference example	2.6.7	2-27
referencing	2.6.6	2-26
ELT processor	1.4.5	1-8
@@END	8.1.1.3.1	8-4
	8.2.1.5	8-12
	A.2	A-8
DCT 1000 or UNISCOPE 100/200	8.2.2.3	8-18
@@END CDI		
DCT 1000	8.2.2.3	8-19
@@END ESC		
DCT 1000 or UNISCOPE 100/200	8.2.2.3	8-18
@@END FUL		
UNISCOPE 100/200	8.2.2.3	8-17
@@END POC		
UTS 400	8.2.2.3	8-19
@@END PTI		
DCT 1000	8.2.2.3	8-19
@ENDCL	3.6.5	3-35
@ENDF	3.8.2	3-66
@EOF	3.4.4.3	3-20
Equipment codes	Appendix E	
@@ERLG	A.2	A-9
UTS 400	8.2.2.3	8-20
Error, parity	9.10	9-37
Error codes		
checkpoint/restart	11.4	11-7
handling	4.1.4	4-2
line terminal contingencies	9.9	9-37
mode status codes	C.3	C-19
symbiont output file recovery	5.8	5-29
termination considerations	4.9.2.1	4-55
types	Table 4-3	4-55
ERRPR$	4.10.5	4-82
ERR$	4.3.2.2	4-8
@@ESC	8.1.1.3.1	8-4
	A.2	A-9
DCT 1000 or UNISCOPE 100/200	8.2.2.3	8-19
ESI		
activities	9.5	9-32
activity concept	10.6.1	10-11
activity exit	9.5	9-33
contingencies	4.9.5	4-66
exiting from an activity	9.5	9-33
interrupts	10.6.2.1	10-11
processing	12.5.6.1	12-19
real-time concepts	10.6.2.2	10-12
timing	10.6.2	10-11
Executive		
addressing windows	12.3.1.4	12-7
basic design philosophy	12.2	12-1

Term	Reference	Page
cataloged file recovery	12.8	12-25
control language	1.3.3.1	1-4
control statements	Section 3 Appendix A	
general	1.3	1-2
interlock processing	12.5.6.1	12-19
internal design	Section 12	
interrupt handling	12.7	12-22
main storage usage	12.3	12-3
nonresident components	12.3.3	12-9
PCT usage	12.3.2	12-8
residency of components	12.3.3	12-8
scheduling	12.5	12-10
service requests	Section 4	
Executive Request (ER)	2.5.4 Section 4	2-20
ABORT$	4.3.2.3	4-8
ABSAD$	4.7.4	4-27
ACSF$	4.10.1.2	4-76
ACT$	4.3.3.4	4-11
ADACT$	9.5	9-33
ADED$	4.3.1.3	4-7
APCHCA$	5.4.8	5-21
APCHCN$	5.4.6	5-20
APNCHA$	5.3.8	5-13
APRINT$	5.3.2	5-9
APRNTA$	5.3.4	5-10
APRTCA$	5.4.4	5-20
APRTCN$	5.4.2	5-19
APUNCH$	5.3.6	5-12
AREADA$	5.2.4	5-6
AREAD$	5.2.2	5-4
ATREAD$	5.2.6	5-8
AWAIT$	4.3.3.1	4-10
BANK$	4.8.4	4-40
basic I/O	6.1.1	6-1
BDSPT$	4.7.6	4-29
CADD$	9.4.2.3	9-29
calling sequence conventions	4.1.2	4-1
CEND$	4.9.4.2	4-63
CGET$	9.4.2.2	9-29
CJOIN$	9.4.2.4	9-30
CLIST$	5.5	5-22
CMD$	9.4.1.2	9-16
CMH$	9.4.1.9	9-21
CMI$	9.4.1.3	9-17
CMO$	9.4.1.4	9-18
CMSA$	9.4.1.5	9-18
CMS$	9.4.1.1	9-15
CMT$	9.4.1.10	9-21
coding restrictions	4.1.1	4-1
COM$	4.6.1	4-22
COND$	4.4.2	4-20
CPOOL$	9.4.2.1	9-26
CQUE$	4.9.6.3	4-70
CREG$	4.9.3.2	4-61
CREL$	9.4.2.5	9-30
CRTN$	4.9.4.3	4-63
CSF$	4.10.1.1	4-74
C$TS	4.3.4.4	4-15
C$TSA	4.3.4.6	4-16
C$TSQ	4.3.4.5	4-15
DACT$	4.3.3.3	4-10
DATE$	4.5.1	4-21
EABT$	4.3.2.4	4-9
EDJS$	4.8.6	4-48
ERRPR$	4.10.5	4-82
ERR$	4.3.2.2	4-8
EXIT$	4.3.2.1	4-8
FACIL$	7.2.7	7-16
FACIT$	7.2.7	7-16
FITEM$	7.2.6	7-4
FORK$	4.3.1.1	4-6
IALL$	4.9.3.1	4-59
IDENT$	4.3.3.6	4-12
II$	4.6.2	4-24
INFO$	4.8.7	4-49
INT$	4.3.3.5	4-11
IOARB$	6.8.2	6-40
IOAXI$	6.8.3	6-41
IOI$	6.3.5	6-11
IOWI$	6.3.7	6-11
IOW$	6.3.6	6-11
IOXI$	6.3.8	6-12
IO$	6.3.4	6-10
I$OD	6.2.2	6-8
I$OT	6.2.1	6-6
LABEL$	7.3.4	7-26
LCORE$	4.7.2	4-25
LEVEL$	4.3.7	4-19
LOAD$	4.7.5	4-28
LOG$	4.8.8	4-53
MCORE$	4.7.1	4-25
MCT$	4.8.3	4-36
NAME$	4.3.3.2	4-10
NRT$	4.3.5.2	4-19
OPT$	4.8.1	4-34
PCHCA$	5.4.7	5-21
PCHCN$	5.4.5	5-20
PCT$	4.8.2	4-34

Term	Reference	Page
PNCHA$	5.3.7	5-12
PRINT$	5.3.1	5-8
PRNTA$	5.3.3	5-9
PRTCA$	5.4.3	5-19
PRTCN$	5.4.1	5-14
PSR$	4.10.2.3	4-78
PUNCH$	5.3.5	5-11
READA$	5.2.3	5-5
READ$	5.2.1	5-3
ROUTE$	9.4.3	9-31
RSI$	8.7	8-36
RT$	4.3.5.1	4-18
SETBP$	4.10.4	4-80
SETC$	4.4.1	4-20
SMU$	4.7.7	4-32
SNAP$	4.10.3	4-79
summary of available	Table 4-1 Appendix A	4-3
SYMB$	5.6	5-24
synchrony	4.1.3	4-2
SYSBAL$	4.8.5	4-44
TDATE$	4.5.2	4-21
TFORK	4.3.1.2	4-7
TIME$	4.5.3	4-22
TINTL$	7.2.8	7-16
TLBL$	7.3.3	7-20
TREAD$	5.2.5	5-7
TRMRG$	4.9.6.5	4-72
TSQCL$	4.3.4.3	4-15
TSQRG$	4.3.4.2	4-14
TSWAP$	7.2.9	7-17
TWAIT$	4.3.6	4-19
T$CELL	4.3.4.1	4-14
UNLCK$	6.3.9	6-12
WAIT$	6.3.1	6-9
WALL$	6.3.3	6-10
WANY$	6.3.2	6-10
with ASCII image	4.10.1.2	4-76
with Fieldata image	4.10.1.1	4-74
within common banks	4.11.5	4-89
Executive Requests within common banks		
CMS$ and CPOOL$	4.11.5.3	4-90
IO$	4.11.5.5	4-90
LOAD$	4.11.5.4	4-90
MCORE$ and LCORE$	4.11.5.1	4-89
EXIT$	4.3.2.1	4-8
real-time activity	10.4.2.4	10-8
release ESI activity control	9.5	9-32
External filename	2.6.2	2-23

F

Term	Reference	Page
Facility		
control statements	3.7	3-37
CSF$ status codes	Table C-1	C-16
inventory and selection	12.5.2	12-11
request status codes	C.2	C-16
status bits	Table C-1	C-16
Facility assignment	1.3.3.3	1-5
alternate method of retrieving	7.2.7	7-16
retrieving (FITEM$)	7.2.6	7-4
FACIL$	7.2.7	7-16
FACIT$	7.2.7	7-16
Fieldata		
BCD translations	Table 6-3	6-22
control statement listing	5.5	5-22
dynamic request of control statements	4.10.1.1	4-74
time and date in	4.5.1	4-21
to ASCII conversion	Table D-1	D-2
Fieldata control functions		
alternate print file	5.4.3	5-19
alternate punch file	5.4.7	5-21
print file	5.4.1	5-14
punch file	5.4.5	5-20
Fieldata images		
alternate file	5.2.3	5-5
alternate print file	5.3.3	5-9
alternate punch file	5.3.7	5-12
conversational mode	5.2.5	5-7
printing	5.3.1	5-8
punching	5.3.5	5-11
reading	5.2.1	5-3
File		
addressing	7.2.3	7-3
administration processor	1.4.6	1-8
assignment	3.7.1	3-37

Term	Reference	Page
attaching internal names	3.7.5	3-63
cataloged, recovery	12.8	12-25
control	1.3.3.4 Section 7	1-5
creation of card image	3.8.1	3-65
cycles (F-cycles)	2.6.3	2-24
data files from NTR sites	3.8.3	3-66
exclusive use	7.2.4	7-3
external and internal names	2.6.2	2-23
independent cataloging	3.7.3	3-58
names	2.6.1	2-22
organization	7.2	7-1
reference example	2.6.7	2-27
referencing	2.6.6	2-26
releasing	3.7.4	3-60
rollout and rollback	7.2.5	7-3
sector-formatted assignment	3.7.1.1	3-39
specifying filename qualifier	3.7.6	3-64
symbiont concepts	2.4.3	2-17
symbiont output queuing	3.6.3	3-31
terminating mode	3.8.2	3-66
@FILE	3.8.1	3-65
File, alternate		
ASCII images	5.2.4	5-6
Fieldata images	5.2.3	5-5
File, alternate print		
ASCII control functions	5.4.4	5-20
ASCII images	5.3.4	5-10
Fieldata control functions	5.4.3	5-19
Fieldata images	5.3.3	5-9
File, alternate punch		
ASCII control functions	5.4.8	5-21
ASCII images	5.3.8	5-13
Fieldata control functions	5.4.7	5-21
Fieldata images	5.3.7	5-12
File, cataloged		
recovery	12.8	12-25
File, magnetic tape		
assigning	3.7.1.2	3-47
initialization	7.2.8	7-16
swapping	7.2.9	7-17
File, print		
ASCII control functions	5.4.2	5-19
Fieldata control functions	5.4.1	5-14
File, punch		
ASCII control functions	5.4.6	5-20
Fieldata control functions	5.4.5	5-20
@FIN	3.4.2	3-15
FITEM$	7.2.6	7-4
equipment codes	Table 7-1	7-13
FITEM$ packet format		
communications peripherals	7.2.6.5	7-11
disk peripherals	7.2.6.6	7-15
magnetic tape peripherals	7.2.6.3	7-8
sector-formatted mass storage	7.2.6.2	7-7
unit record and nonstandard peripherals	7.2.6.1	7-6
word-addressable mass storage	7.2.6.4	7-9
FORK$	4.3.1.1	4-6
FORTRAN, example	F.4	F-5
@FREE		
options	Table 3-7	3-59
status code	Table C-1	C-16
used with @ASG	3.7.1	3-37
@@FUL	A.2	A-9
UNISCOPE 100/200	8.2.2.3	8-17
Function		
EACQ$	6.5.3	6-29
mode set (MS$)	6.4.1.2	6-18

Term	Reference	Page
set mode (SM$)	6.4.1.1	6-15
Function string interpretation	6.8.1.2	6-36
Function (EF) handling	6.8.1.5	6-36
Functions, handler		
disk	6.7	6-35
sector-formatted mass storage	6.6	6-32
tape	6.4.1	6-13
word-addressable storage	6.5.1	6-26
FURPUR processor	1.4.2	1-7
F-cycle	2.6.3	2-24
G		
General Communications Subsystem (GCS)	9.1.1.3	9-2
Guard mode status	C.6	C-38
H		
Handler		
arbitrary device	6.9	6-46
disk	6.7	6-35
disk, free format	6.8.4	6-41
magnetic tape	6.4	6-13
MSA tape arbitrary device	6.8.4	6-41
sector-formatted mass storage	6.6	6-32
word-addressable mass storage	6.5	6-26
Hardware fault interrupts	12.7.3	12-23
@HDG	3.6.1	3-28
Heading, output control	3.6.1	3-28
High Volume Time Sharing (HVTS)	1.4.11	1-9
@@HOLD	8.1.1.3.1 A.2	8-4 A-9
I		
IALL$	4.9.3.1	4-59
for real-time	10.5	10-10
IDENT$	4.3.3.6	4-12
Idle line monitor	9.6.1	9-33
Idle line polling	9.6.2	9-34
II$	4.6.2	4-24
Images, ASCII		
alternate file	5.2.4	5-6
alternate print file	5.3.4	5-10
alternate punch file	5.3.8	5-13
printing	5.3.2	5-9
punching	5.3.6	5-12
reading	5.2.2	5-4
Images, Fieldata		
alternate file	5.2.3	5-5
alternate print file	5.3.3	5-9
alternate punch file	5.3.7	5-12
conversational mode	5.2.5	5-7
printing	5.3.1	5-8
punching	5.3.5	5-11
reading	5.2.1	5-3
INFO$	4.8.7	4-49
Input		
paper tape	8.2.1.2.2	8-10
solicated console	4.6.1	4-22
unsolicited console	4.6.2	4-24
Input/output		
basic Executive Request	6.1.1	6-1
codes defined in SYS$*RLIB$	Table 6-1	6-4
device handlers	Section 6	
device handlers and symbionts	1.3.3.6	1-5
disconnecting channel	6.8.1.4	6-40
interrupt handling	12.7.1	12-22
magnetic tape packet generation (I$OT)	6.2.1	6-6

Term	Reference	Page
mass storage packet generation (I$OD)	6.2.2	6-8
packet format	Figure 6-1	6-2
packet generation	6.2	6-6
priority, real-time	10.4.1.1	10-5
program synchronization	6.3	6-9
status codes	Table C-2	C-19
wait for completion of any	6.3.2	6-10
	6.3.3	6-10
wait for completion of specific	6.3.1	6-9
Input/output initiation		
exit, with interrupt	6.3.8	6-12
return control immediately	6.3.4	6-10
return control immediately, with interrupt	6.3.5	6-11
wait for completion	6.3.6	6-11
wait for completion, with interrupt	6.3.7	6-11
@@INQ	8.1.1.3.1	8-5
	A.2	A-9
@@INS	A.2	A-9
UNISCOPE 100/200	8.2.2.3	8-17
Interface		
arbitrary device	6.9	6-46
demand symbiont	8.1.1.3	8-3
routines, symbiont user	5.1.2	5-2
Interlock processing	12.5.6.1	12-19
Internal filename	2.6.2	2-23
Interprocessor interrupt	12.7.2	12-23
Interrupt		
activity	6.1.2	6-5
activity priority reduction	6.3.9	6-12
handling	12.7	12-22
hardware fault	12.7.3	12-23
input/output	12.7.1	12-22
interprocessor	12.7.2	12-23
inter-activity	4.3.3.5	4-11
power loss	12.7.3.2	12-24
program generated	12.7.4	12-24
real-time console handling	10.4.2.6	10-9
reducing activity priority	6.3.9	6-12
response	9.7.1	9-34
storage and ICR parity error	12.7.3	12-23
Interrupt activity handling	6.8.1.6	6-36
Interrupt handling		
interprocessor interrupts	12.7.2	12-23
I/O interrupts and queuing	12.7.1	12-22
power loss	12.7.3.2	12-24
program generated interrupts	12.7.4	12-24
storage and ICR parity error interrupts	12.7.3	12-23
INT$	4.3.3.5	4-11
IOARB$	6.8.2	6-40
IOAXI$	6.8.3	6-36
IOI$	6.3.5	6-11
IOWI$	6.3.7	6-11
IOW$	6.3.6	6-11
IOXI$	6.3.8	6-12
IO$	6.3.4	6-10
I$OD	6.2.2	6-8
I$OT	6.2.1	6-6
J		
@JUMP	3.10.4.3	3-76

Term	Reference	Page
L		
Label field	3.2.1	3-3
Labeling		
control statements	3.10.3	3-72
disk	7.4	7-28
tape	7.3	7-18
LABEL$	7.3.4	7-26
Language processor, control statements	3.9	3-67
LCORE$	4.7.2	4-25
restrictions	4.7.3	4-26
usage within common banks	4.11.5.1	4-89
LEVEL$	4.3.7	4-19
Library		
operating system files	3.9	3-67
relocatable subroutine	1.6	1-9
LIB$	3.9	3-67
LIJ/LBJ/LDJ, switching between banks	3.4.4.4.2	3-22
Line terminal		
deactivation of input/output	9.4.1.10	9-21
device assignment	9.2	9-4
group initialization	9.4.1.1	9-15
table input status codes	Table 9-1	9-6
LIST processor	1.5.3	1-9
Listing, user-defined control statement	5.5	5-22
LOAD$		
loading, direct	4.7.5	4-28
usage with common banks	4.11.5.4	4-90
@LOG	3.5.2	3-27
Logging on, TSS		
basic mode	8.3.2.1	8-22
batch run	8.3.2.2	8-23
run mode	8.3.2.3	8-23
LOG$	4.8.8	4-53
LPD (Load Processor Designators)	4.10.2.2	4-77
LTT input, status codes	Table 9-2	9-9
LTT output, status codes	Table 9-1	9-6
M		
Magnetic tape		
assignment	3.7.1.2	3-47
devices	1.3.3.6	1-6
FITEM$ request packet	7.2.6.2	7-7
functions vs. unit type	Table 6-4	6-25
handler functions	6.4.1	6-13
I/O function with interrupt	6.2.1.2	6-7
I/O function without interrupt	6.2.1.1	6-7
I/O functions and codes	Table 6-2	6-14
noise constant	6.4.2.1	6-20
peripherals	7.2.6.3	7-8
Main storage		
absolute addressing	4.7.4	4-27
allocation	2.5.2	2-19
contraction	4.7.2	4-25
dynamic allocation	12.5.5.2	12-16
Executive components	12.3.3	12-8
expansion	4.7.1	4-25
for buffers	10.3.2	10-2
layout, 1100/80	12.3.1.3	12-6
layout, 1106, 1108, 1100/10/20	12.3.1.1	12-3
layout, 1110, 1100/40	12.3.1.2	12-5
snapshot dump	4.10.3	4-79

Term	Reference	Page
Mass storage		
allocation	7.2.2	7-2
assigning word-addressable	3.7.1.3	3-54
devices	1.3.3.6	1-6
disk	6.7	6-35
I/O function with interrupt	6.2.2.2	6-9
I/O function without interrupt	6.2.2.1	6-8
I/O packet generation	6.2.2	6-8
utilization	1.3.2	1-3
Master configuration table (MCT), retrieval of	4.8.3	4-36
Master File Directory	7.2.1	7-1
Master log, inserting information	3.5.2	3-27
MCORE$	4.7.1	4-25
restrictions	4.7.3	4-26
usage within common banks	4.11.5.1	4-89
MCT$	4.8.3	4-36
Message		
control statements	3.5	3-25
displaying	3.5.1	3-25
@MODE	3.7.2	3-57
status code	C.2	C-16
	C.3	C-19
Monitor handling	6.8.1.3	6-36
@MSG	3.5.1	3-23
options	Table 3-3	3-24
Multiprocessing	12.4	12-10
general	1.3.1.5	1-3
Multiprogramming		
application to real-time	10.4.2.2	10-7
considerations	2.5.5	2-20
general	1.3.1.5	1-3

N

Term	Reference	Page
NAME$	4.3.3.2	4-10
Non Configured Common Banks (NCCB)	4.11.1.1.2	4-86
@@NOPR	A.2	A-9
UNISCOPE 100/200	8.2.2.3	8-17
Notational conventions	2.3.1	2-15
NRT$	4.3.5.2	4-19
real-time	10.4.2.1	10-7

O

Term	Reference	Page
Octal/decimal conversion	Table D-5	D-8
Omnibus element	2.6.4	2-25
Operand fields	3.2.3	3-3
Operating system	1.2	1-1
library files	3.9	3-67
Operation		
demand terminal modes	8.1.1.2	8-3
fields	3.2.2	3-3
Operator Communications	1.3.3.5	1-5
OPT$	4.8.1	4-34
Output		
print heading control	3.6.1	3-28
symbiont file, queuing	3.6.3	3-31

P

Term	Reference	Page
Paper tape operations		
input	8.2.1.2.2	8-10
output	8.2.1.2.1	8-10
Parity errors	9.10	9-37
@@PASSWD	8.3.2.6	8-25

Term	Reference	Page
Password modification	8.3.2.6	8-25
PCHCA$	5.4.7	5-21
PCHCN$	5.4.5	5-20
PCT		
referencing	3.4.4.4.6	3-24
usage	12.3.2	12-8
PCT$	4.8.2	4-34
PDP processor	1.4.8	1-8
Peripheral devices		
assignment	3.7.1	3-37
releasing	3.7.4	3-60
Peripherals, FITEM$ request packet		
communications	7.2.6.5	7-11
disk	7.2.6.6	7-15
magnetic tape	7.2.6.3	7-8
nonstandard	7.2.6.1	7-6
sector-formatted mass storage	7.2.6.2	7-7
PMD processor	1.4.3	1-8
@@PMOD	A.2	A-9
UTS 400	8.2.2.3	8-20
PNCHA$	5.3.7	5-12
@@POC	A.2	A-10
UTS 400	8.2.2.3	8-20
Pool		
dual method for input	9.4.1.8	9-21
dual method for real-time	10.3.5	10-4
mode for I/O operations	9.4.1.7	9-19
size for real-time	10.3.3	10-3
Postmorten dump, options	3.4.1.1	3-13
Power loss interrupt	12.7.3.2	12-24
Print		
control functions	Table 5-2	5-15
output heading control	3.6.1	3-28
Print file		
ASCII control functions	5.4.2	5-19
Fieldata control functions	Table 5-2	5-15
Print file, alternate		
ASCII control functions	5.4.4	5-20
ASCII images	5.3.4	5-10
Fieldata control functions	5.4.3	5-19
Fieldata images	5.3.3	5-9
Printing		
ASCII images	5.3.2	5-9
Fieldata images	5.3.1	5-8
PRINT$	5.3.1	5-8
Priority		
changing, real-time	10.4.2.1	10-7
control real-time	10.4.2	10-7
dispatching for real-time	10.4.1.2	10-5
interrupt activity reduction	10.4.2.3	10-8
I/O for real-time	10.4.1.1	10-5
reducing interrupt activity	6.3.9	6-12
@@PRNT	A.2	A-10
DCT 1000	8.2.2.3	8-19
UNISCOPE 100/200	8.2.2.3	8-17
PRNTA$	5.3.3	5-9
Procedure definition processor (PDP)	1.4.8	1-8
Processing		
buffer	9.7.2	9-35
contingency	4.9.4	4-62
demand	Section 8	
interlock	12.5.6.1	12-19
real-time	1.3.1.3	1-3
	Section 10	
Processor		
control statements	3.9	3-67

Term	Reference	Page
dedication	4.3.1.3	4-7
distributed communications	9.1.3	9-3
language	3.9	3-68
system	1.4	1-7
system utility	1.5	1-9
use of SI and SO parameters	Table 3-9	3-68
use of SI, SO, and RO parameters	Table 3-8	3-67
use of TSS	8.3.3	8-26
Processor State Register (PSR)		
altering/retrieving	4.10.2	4-76
load	4.10.2.2	4-77
PSR$	4.10.2.3	4-78
store	4.10.2.1	4-76
Program		
abortion	4.3.2.3	4-8
bank referencing	3.4.4.4	3-21
changing to real-time status	4.3.5.1	4-18
control	4.3	4-6
control table retrieval	4.8.2	4-34
control table referencing	3.4.4.4.6	3-24
data separation	3.4.4.3	3-20
error termination	4.3.2.4	4-9
initial activity	3.4.4.2	3-19
initial status	3.4.4.1	3-18
initiating execution	3.4.4	3-18
I/O synchronization	6.3	6-9
protection	2.5.7	2-21
real-time location	10.2	10-1
removing real-time status	4.3.5.2	4-19
storage control	4.7	4-25
termination	3.4.4.5	3-25
Program execution		
common bank access	3.4.4.2.4	3-20
initially based common banks	3.4.4.2.3	3-20
initiating	3.4.4	3-18
lowest bank address	3.4.4.2.2	3-19
overlapped addresses	3.4.4.2.1	3-19
Program-generated interrupts	12.7.4	12-24
PRTCA$	5.4.3	5-19
PRTCN$	5.4.1	5-14
PSR$	4.10.2.3	4-78
@@PTI	A.2	A-10
DCT 1000	8.2.2.3	8-19
@@PTO	A.2	A-10
DCT 1000	8.2.2.3	8-19
@@PTP	A.2	A-10
DCT 1000	8.2.2.3	8-19
Punch file		
ASCII control functions	5.4.6	5-20
Fieldata control functions	5.4.5	5-20
Punch file, alternate		
ASCII control functions	5.4.8	5-21
Fieldata control functions	5.4.7	5-21
Fieldata images	5.3.7	5-12
Punching		
ASCII images	5.3.6	5-12
Fieldata images	5.3.5	5-11
PUNCH$	5.3.5	5-11
Q		
@QUAL	3.7.6	3-64
Quantum timer	12.6.3	12-22
Queuing	6.1.3	6-5
R		
Reading		
ASCII images	5.2.2	5-4
backward limitations	6.4.2.2	6-20
Fieldata images	5.2.1	5-3
tape label blocks	7.3.3	7-20
READA$	5.2.3	5-5

Term	Reference	Page
READ$, bit settings returned in A0	Table 5-1	5-4
Real-time	Section 10	
clock	12.6.1	12-22
general	1.3.1.3	1-3
tasks	2.5.1	2-19
Real-time buffer operations		
buffer size	10.3.4	10-3
dual pool method	10.3.5	10-4
main storage availability	10.3.2	10-2
pool size	10.3.3	10-3
transmission types	10.3.1	10-2
Real-time priority		
activity termination	10.4.2.4	10-8
application of multiprogramming	10.4.2.2	10-7
changing activity	10.4.2.1	10-7
console interrupt handling	10.4.2.6	10-9
control	10.4.2	10-7
dispatching	10.4.1.2	10-5
I/O	10.4.1.1	10-5
priority reduction	10.4.2.3	10-8
timed wait considerations	10.4.2.5	10-8
Real-time processing		
buffer operations	10.3	10-2
ESI considerations	10.6	10-11
introduction	10.1	10-1
program execution considerations	10.4	10-5
program location	10.2	10-1
program responsibilities	10.5	10-10
Register basing	4.11.3.2	4-89
Relocatable element	2.6.4	2-25
Relocatable subroutines library	1.6	1-9
Restart		
contingencies	11.3.4	11-4
control statement	11.3.1	11-2
error codes	Table 11-3	11-7
error messages	Table 11-4	11-9
Executive Request	11.3.2	11-4
file handling	11.3.5	11-6
initializing	11.3.3	11-4
introduction	11.1	11-1
@RSTRT	11.3.1	11-2
@@RLD	A.2	A-10
UNISCOPE 100/200	8.2.2.3	8-17
@@RLU	A.2	A-10
UNISCOPE 100/200	8.2.2.3	8-17
Rollback	7.2.5	7-3
Rollout	7.2.5	7-3
ROUTE$	9.4.3	9-31
@@RQUE	8.1.1.3.1	8-5
	A.2	A-10
RSI$	8.7	8-36
output functions	Table 8-3	8-41
@RSTRT	11.3.1	11-2
RT$		
real-time	4.3.5.1	4-18
	10.4.2.1	10-7
	10.5	10-10
Run		
abort	4.3.2.3	4-8
branching from within a stream	3.10.4.3	3-76
demand example	8.6	8-34
diagnostic messages	C.1	C-1
dynamic initiation	3.4.3	3-16
example	F.8	F-16
execution	2.4.2	2-17
	3.4.2	3-15
initiation	2.4.1	2-17
	3.4.1	3-9
recovery, R option	3.4.1.2	3-14
setup examples	Appendix F	
stream expansion	3.10.1	3-70
termination	2.4.4	2-18
	3.4.2	3-15
@RUN		
example	3.4.1.3	3-14
for real-time	10.5	10-10
options	Table 3-2	3-8

Term	Reference	Page
Runstream		
branching from within	3.10.4.3	3-76
diagnostic messages	C.1	C-1
example	3.10.4.4	3-77
expansion	3.10.1	3-70
S		
Scatter/gather		
UNISERVO 20	6.4.2.6	6-23
@@SCBY	A.2	A-10
UTS 400	8.2.2.3	8-20
Scheduling		
coarse scheduler	12.5.4	12-13
control statement interpreter	12.5.3	12-12
dispatcher	12.5.6	12-19
dynamic allocator	12.5.5	12-15
facilities inventory and selection	12.5.2	12-11
general	12.5.1	12-10
Sector-formatted mass storage		
file assignment	3.7.1.1	3-39
FITEM$ request packet	7.2.6.2	7-7
I/O functions and codes	Table 6-6	6-32
SECURE processor	1.4.6	1-8
@@SEND	8.1.1.3.1	8-5
	A.2	A-10
Series 600 tape cassette systems	8.2.2.1	8-15
SETBP$	4.10.4	4-80
@SETC	3.10.4.1	3-74
SETC$	4.4.1	4-20
SIP	4.8.5	4-44
@@SKIP	8.1.1.3.1	8-5
	A.2	A-10
SMU$	4.7.7	4-32
Snapshot dump, main storage	4.10.3	4-79
SNAP$	4.10.3	4-79
SPD, store processor designators	4.10.2.1	4-76
SSG processor	1.4.9	1-8
@START	3.4.3	3-16
status codes	Table C-4	C-37
Status, initial execution	3.4.4.1	3-18
Status codes		
CSF$	C.4	C-35
ERR mode	C.3	C-18
facility request	C.2	C-16
	C.4.1	C-35
guard mode and exception status for 1100/80	C.6	C-38
@ADD	C.4.3	C-36
@BRKPT	C.4.2	C-35
@START	C.4.4	C-36
@SYM	C.4.2	C-35
Storage parity error interrupts	12.7.3.1	12-23
SUP formula	12.5.5.2	12-17
Supervisor	1.3.3.2	1-4
Switching	12.5.6.2	12-20
@SYM		
example of usage	3.6.4	3-33
status code	Table C-3	C-35
Symbiont		
primary output file breakpoint	3.6.2.1	3-29
Symbiont demand	8.2	8-9
general operation	8.1.1	8-2

Term	Reference	Page
Symbionts		
alternate file breakpoint	3.6.2.2	3-30
control statements	Table 8-1	8-4
demand	8.2	8-9
demand interface	8.1.1.3	8-3
directive statements	3.6	3-27
file concepts	2.4.3	2-17
general	5.1.1	5-1
input/output	1.3.3.6	1-5
interface requests	Section 5	
output file error recovery	5.8	5-29
output file queuing	3.6.3	3-31
user interface routines	5.1.2	5-2
Symbiont, demand		
DCT 1000	8.2.2	8-13
DCT 500/475	8.2.1	8-9
Tektronix 4013	8.2.1.7	8-13
teletypewriter	8.2.1	8-9
UNISCOPE 100/200	8.2.2	8-13
Symbolic element	2.6.4	2-25
cycles	2.6.5	2-25
SYMB$	5.6	5-24
SYSBAL$	4.8.5	4-44

T

Term	Reference	Page
Table, Collector produced	4.11.3.3	4-89
Tape		
ADI considerations	6.8.5	6-43
assigning files	3.7.1.2	3-47
handler functions	6.4.1	6-13
labeling, TLBL$	7.3.3	7-20
reading and writing label blocks	7.3.3	7-20
swapping reels (TSWAP$)	7.2.9	7-17
unit mode control	3.7.2	3-57
Tape file, example	F.6	F-9
Tape handler		
abnormal frame count considerations	6.4.2.5	6-23
functions	6.4.1	6-13
move considerations	6.4.2.4	6-21
read backward limitations	6.4.2.2	6-20
scatter read/gather write considerations for UNISERVO 20	6.4.2.6	6-23
write considerations	6.4.2.3	6-20
Tape labeling, definitions	7.3.1	7-18
Tape, paper input	8.2.1.2.2	8-10
Task		
batch	12.5.5.1	12-15
control, basic concepts	2.5	2-19
deadline	12.5.5.1	12-15
demand	12.5.5.1	12-15
execution and switching	2.5.3	2-19
initiation	2.5.2	2-19
real-time	2.5.1	2-19
	12.5.5.1	12-15
termination	2.5.6	2-21
transaction	12.5.5.1	12-15
@@TCI	A.2	A-10
UNISCOPE 100/200	8.2.2.3	8-18
@@TCM	A.2	A-10
UNISCOPE 100/200	8.2.2.3	8-18
@@TCO	A.2	A-11
UNISCOPE 100/200	8.2.2.3	8-18
@@TCT	A.2	A-11
UNISCOPE 100/200	8.2.2.3	8-18
TDATE$	4.5.2	4-21
Tektronix Model 4013	8.2.1.7	8-13
Teletypewriter		
demand symbiont	8.2.1	8-9
interrupting output	8.2.1.4	8-11
operation modification control statements	8.2.1.5	8-12
operational considerations	8.2.1.1	8-9
paper tape operations	8.2.1.2	8-10
special characters	8.2.1.3	8-11

Term	Reference	Page
Temporary program file (TPF$)	3.9	3-67
@@TERM	8.1.1.3.1	8-5
	A.2	A-11
Terminal Security System (TSS)		
basic mode	8.3.2.1	8-22
capability	8.4	8-27
contingencies	8.3.2.5	8-25
error messages	8.3.4	8-26
general	8.3.1	8-21
logging on	8.3.2	8-22
	8.3.2.3	8-23
usage	8.3.3	8-26
user-id	8.3.2.7	8-25
@TEST	3.10.4.2	3-75
Test and set		
cell format	4.3.4.1	4-14
clear and activate	4.3.4.6	4-16
clear and notify EXEC	4.3.4.4	4-15
clear and queue	4.3.4.5	4-15
deregistration	4.3.4.3	4-15
examples of usage	4.3.4.7	4-17
protecting data within	4.11.7	4-90
queuing	4.3.4	4-13
	4.11.7.3	4-90
registration	4.3.4.2	4-14
usage	4.11.7.1	4-90
	10.4.3	10-9
Text editor processor (ED)	1.4.7	1-8
TFORK$	4.3.1.2	4-7
Time and date		
binary	4.5.2	4-21
Fieldata	4.5.1	4-21
milliseconds	4.5.3	4-22
Time sharing	1.3.1.2	1-3
	12.5.5.4	12-18
Timed wait		
activity	4.3.6	4-19
considerations	10.4.2.5	10-8
creating activity with	4.3.1.2	4-7
TIME$	4.5.3	4-22
TINTL$	7.2.8	7-16
TIP	1.3.1.4	1-3
TLBL$, errors	7.3.3	7-25
@@TM	8.1.1.3.1	8-5
	A.2	A-11
TPF$	3.9	3-67
Transaction Processing Interface (TIP)	1.3.1.4	1-3
Transparent control statement	3.2.8	3-6
	8.1.1.3.2	8-6
TREAD$	5.2.5	5-7
TRMRG$	4.9.6.5	4-72
@@TSBM	A.2	A-11
UTS 400	8.2.2.3	8-21
TSQCL$	4.3.4.3	4-15
TSQRG$	4.3.4.2	4-14
TSWAP$	7.2.9	7-17
@@TTY	8.2.1.5	8-12
	A.2	A-11
TWAIT$	4.3.6	4-19
T$CELL	4.3.4.1	4-14

U

Term	Reference	Page
UNISCOPE 100/200		
control statements	8.2.2.3	8-17
cursor/SOE	Table D-3	D-6
display terminal	D.4	D-5
operational considerations	8.2.2.1	8-13
UNISERVO VIC and VIIC		
function vs. unit type	Table 6-4	6-25

Term	Reference	Page
unit record FITEM$ request packet	7.2.6.1	7-6
UNLCK$	6.3.9	6-12
real-time	10.4.2.3	10-8
@USE	3.7.5	3-63
status code	C.2	C-16
User, breakpoint setting	4.10.4	4-80
User-id	3.4.1	3-11
TSS	8.3	8-21
UTS 400		
cursor/SOE	Table D-3	D-6
display terminal	D.4	D-5
operational considerations	8.2.2.4	8-21
V		
1100 Virtual Machine Control Processor (FLIT)	1.5.4	1-9
W		
Wait		
completion of any I/O	6.3.2	6-10
	6.3.3	6-10
completion of specific I/O	6.3.1	6-9
initiate ADI for unsolicited interrupt	6.9.4	6-36
WAIT$	6.3.1	6-9
WALL$	6.3.3	6-10
WANY$	6.3.2	6-10
Word-addressable mass storage		
assigning files	3.7.1.3	3-54
considerations	6.5.2	6-26
FITEM$ packet format	7.2.6.4	7-9
functions/codes	Table 6-5	6-27
I/O functions and codes	6.5	6-26
Write		
considerations for tape	6.4.2.3	6-20
protect mode	4.11.2	4-88
Writing, tape label blocks	7.3.3	7-20
X		
@XQT, retrieving options (OPT$)	4.8.1	4-34
@@X	8.1.1.3.1	8-5
	A.2	A-11

Comments concerning the content, style, and usefulness of this manual may be made in the space provided below. Please fill in the requested information.

This User Comment Sheet will not normally lead to a reply to the originator. Requests for copies of manuals, lists of manuals, pricing information, etc. must be made through your Series 1100 site manager, to your Sperry Univac representative, or to the Sperry Univac office serving your locality. Software problems should be submitted on a Software User Report (SUR) form UD1–745. Questions of a technical nature regarding either the manual or the software should be submitted on a Technical Question (question/answer) form UD1–1195. These forms are available through your Sperry Univac representative.

Customer Name: _____ System Type: _____

Title of Manual: _____

UP No.: _____ Revision No.: _____ Update: _____

Name of User: _____ Date: _____

Address of User: _____

Comments: Give page and paragraph reference where appropriate.

Please rate this manual.	Good	Adequate	Not Adequate
Organization of the text			
Clarity of the text			
Adequacy of coverage			
Examples			
Cross references			
Tables			
Illustrations			
Index			
Appearance			

YOUR COMMENTS, PLEASE - - - -

This manual is part of a library that serves as a source of information for personnel using SPERRY UNIVAC® systems. Space is provided on the opposite side of this form for your comments concerning the usefulness of the information presented. Each comment will be carefully reviewed by the persons responsible for writing and publishing this manual. All comments and suggestions become the property of Sperry Univac.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY

SPERRY UNIVAC

ATTN: Systems Support
1100 Systems Publications
M.S. 4533

P.O. Box 43942
St. Paul, Minnesota 55164

CUT

FOLD