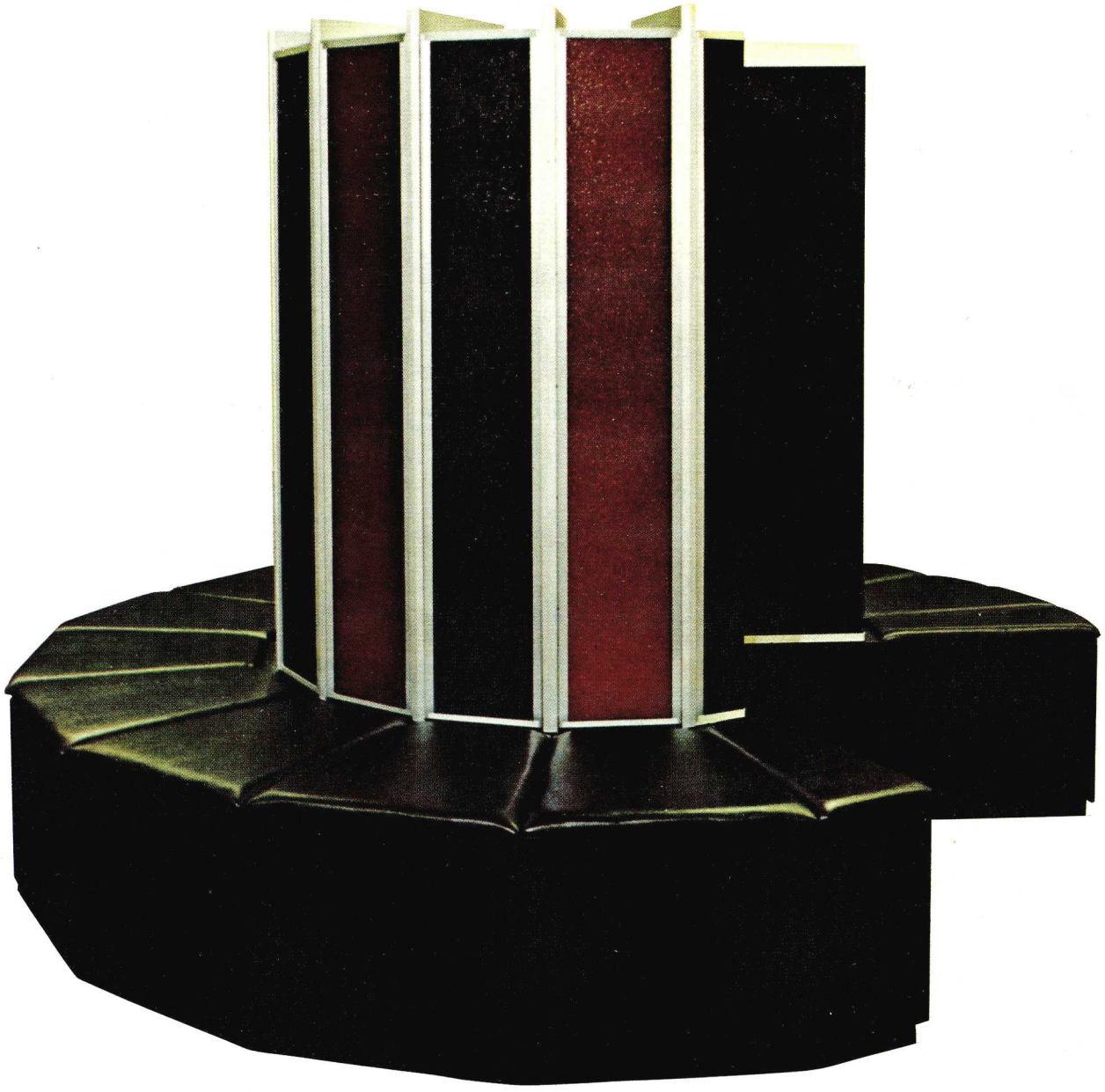
CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEM

CRAY-OS VERSION 1 SYSTEM PROGRAMMER'S MANUAL

2240012



VOLUME TWO

CRAY-1®
COMPUTER SYSTEM

CRAY-OS VERSION 1
SYSTEM PROGRAMMER'S
MANUAL
2240012

VOLUME TWO

PART 2 SYSTEM GENERATION
PART 3 SYSTEM STARTUP AND RECOVERY
PART 4 SYSTEM MODIFICATION AND MAINTENANCE

RECORD OF REVISION

PUBLICATION NUMBER 2240012

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	January 1977 - Original printing
A	May 1977 - Reprint with revision. This revision obsoletes the previous edition.
A-01	August 1977 - Update packet. This packet reflects changes to EXEC, SCP, PFM, EXP, and operator commands. It also provides interim procedures for system generation.
B	January 1978 - Reprint with revision. This revision obsoletes all previous editions. This printing coincides with the release of Version 1 of the CRAY-1 Operating System.
B-01	April 1978 - Update packet. This packet reflects changes incorporated into Version 1.01 of the CRAY-1 Operating System.
B-02	July 1978 - Update packet. This packet reflects changes incorporated into Version 1.02 of the CRAY-1 Operating System.
C	October 1978 - Reprint with revision. This revision obsoletes all previous editions. With this reprint, this publication has been divided into three volumes; Volume 1 contains Part 1 of this publication, Volume 2 includes Parts 2, 3, and 4, and Volume 3 contains Part 5. This printing coincides with the release of version 1.03 of the CRAY-1 Operating System.
C-01	January 1979 - Update packet. This packet reflects changes incorporated into Version 1.04 of the CRAY-1 Operating System.
C-02	April 1979 - Update packet. This packet reflects changes incorporated into Version 1.05 of the CRAY-1 Operating System.
C-03	July 1979 - Update packet. This packet reflects changes incorporated into Version 1.06 of the CRAY-1 Operating System.
D	September 1979 - Reprint. Revision D is the same as Revision C with change packets C-01, C-02 and C-03 added. No additional changes have been made.
D-01	December, 1979 - Update packet. This packet reflects changes incorporated into Version 1.07 of the CRAY-1 Operating System.
D-02	April, 1980 - Update packet. This packet reflects changes incorporated into Version 1.08 of the CRAY-1 Operating System.

E	May, 1980 - Reprint. Revision E is the same as Revision D with change packets D-01 and D-02 added. No additional changes have been made.

PREFACE

The System Programmer's Manual is written for programmers, analysts, and field engineers who are responsible for installing, debugging, or modifying the CRAY Operating System, Version 1.

This manual contains information to aid the programmer in making the transition from the external features of the operating system as described in the CRAY-OS Version 1 Reference Manual, CRI publication SR-0011 to the listings. The reader is assumed to be familiar with the contents of the CRAY-OS Reference Manual and to be experienced in coding in the CRAY Assembly Language (CAL) as described in the CAL Version 1 Reference Manual, CRI publication SR-0000.

Although a general familiarity with the concept of operating systems is assumed, this publication does not presume that the reader knows the principles or techniques of any other specific operating system.

This manual is in three parts, as follows:

VOLUME ONE

PART 1 SYSTEM COMPONENTS

This part familiarizes the reader with the structure, major components, interfaces, and philosophy of the system.

VOLUME TWO

PART 2 SYSTEM GENERATION

This part describes the Cray Research released software materials and how they are used to bring the CRAY-OS Operation System and its product set to an operational state.

PART 3 SYSTEM STARTUP AND RECOVERY

This part gives the procedure for installing, starting, or recovering the operating system.

PART 4 SYSTEM MODIFICATION AND MAINTENANCE

This part gives the rules and conventions to be followed when modifying or adding to the system. It also describes system macros, techniques for adding features to the system, and tools and techniques for analyzing and diagnosing problems and failures.

VOLUME THREE

PART 5 SYSTEM TABLE DESCRIPTIONS

This section contains detailed descriptions of tables resident in EXEC and STP.

Part 2

SYSTEM GENERATION

CONTENTS
PART 2 SYSTEM GENERATION

1. CRI SOFTWARE INITIALIZATION 1.1-1

1.1 CRI SOFTWARE RELEASE MATERIALS 1.1-1

1.1.1 UPDATE program libraries 1.1-2

1.1.2 General descriptions of release tape contents . . 1.1-3

1.2 INITIALIZATION PROCEDURE 1.2-1

2. COS MODIFICATION 2.1-1

2.1 INSTALLATION DEPENDENT MODIFICATIONS 2.1-1

2.1.1 Installation parameters 2.1-1

2.1.2 Hardware parameters 2.1-7

2.2 LOCAL SYSTEM GENERATION 2.2-1

3. SAMPLE UPDATE JOB 3.1-1

3.1 COSGEN JOB 3.1-1

4. DGS STATION GENERATION 4.1-1

4.1 PROCEDURE 4.1-1

4.2 LISTINGS 4.2-1

4.2.1 GENERATE.MC 4.2-1

4.2.2 UTILITY.MC 4.2-4

4.2.3 TAPE.MC 4.2-6

4.2.4 LISTING.MC 4.2-7

CRI SOFTWARE INITIALIZATION 1

This section describes the Cray Research released software materials and how they are used to bring the CRAY-OS Operating System and its product set to an operational state. Other sections in Part 2 tell how to modify or configure the software according to a local site's preferences.

1.1 CRI SOFTWARE RELEASE MATERIAL

Software for COS is delivered to a customer as a set of nine tapes. The documentation accompanying these materials includes change packets, errata, or addenda for external and internal documentation, if required; a software problem report summary; a modification report describing each modification included in the release; and a software release letter summarizing the current status of the software. If any critical modifications need to be made to the release software, the modifications will also be included.

The released software is configured for a minimum CRAY-1 hardware configuration (one disk drive and one-quarter million words of memory). Using this system, local modifications in the form of UPDATE input can be written and applied to the source program libraries to generate a system meeting local hardware configuration and software preferences.

The nine tapes are identified as:

1. RDOS
2. COSPL
3. COSPL listings
4. PRODPL
5. PRODPL listings
6. CFTPL with listings
7. FTLIBPL with listings
8. DGS software
9. DGS listings

1.1.1 UPDATE PROGRAM LIBRARIES

Four of the release tapes (COSPL, PRODPL, CFTPL, and FTLIBPL) contain UPDATE program libraries. Decks in these libraries comprise all CRI software that runs on the CRAY-1. In the following, decks that must be modified and reassembled to generate a new binary for a program are listed in the left columns. Datasets affected by the change are listed in the right columns. (Common decks are not included in this list.)

COSPL		PRODPL		CFTPL		FTLIBPL
Deck	Dataset	Deck	Dataset	Deck	Dataset	
ST	$SYSTXT	C	CAL	F	CFT	There is a separate deck name for each program within the $FTLIB dataset
CT	COSTXT	L	LDR			
E	EXEC	UF, UC	UPDATE			
S	STP					
J	CSP					
AF, AC	AUDIT	CSF, CSC	CSIMR			
CD	COPYD	BF	BUILD			SCILBPL
CF	COPYF					There is a separate deck name for each program within the $SCILIB dataset. (SCILBPL and $SCILIB are on the FTLIBPL tape.)
CR	COPYR	FDF, FDC	FDUMP			
DD	DSDUMP					
DM	DUMP	SKOL	SKOLTXT, SKOL			
EXF, EXC	EXTRACT					
JCF, JCC	JCSDEF					
PD	PDSDUMP					SYSLBPL
PL	PDSLOAD					SYSLBPL and $SYSLIB are on the FTLIBPL tape.
SK	SKIPD, SKIPF, SKIPR					
UN	UNB					
WD	WRITEDS					
CM, CMC	COMPARE					
DDC	DDC					
DBF	DEBUG					
STATS	STATS					

To obtain a listing of all deck names in a program library, run UPDATE with the ID list option selected.

1.1.2 GENERAL DESCRIPTIONS OF RELEASE TAPE CONTENTS

General descriptions of the contents of the release tapes are given in this subsection. These descriptions are not all-inclusive but highlight the tape contents.

RDOS

This tape contains Data General-written and CRI-written software binaries for execution on the ECLIPSE minicomputer system. The software includes the RDOS Operating System and the operator station for the CRAY-1.

File	Contents
0	TBOOT.SV, the tape boot program
1	Required programs for running RDOS
2	SYS.SV, the RDOS operating system save file
3	SYS.OL, the RDOS operating system overlay file
4	DKINIT.SV, the disk initializer program
5	BOOT.SV, the disk boot program
6	Other Data General-written programs executing under RDOS
7	Files for generating RDOS systems
8	Files for generating RDOS systems
9	CRI-written programs executing under RDOS (including DGS binary)
10	DPDF.SV, the disk pack formatter program

COSPL

This tape contains all of the software associated with the COSPL program library. In addition, it contains some software common to the entire operating system.

File	Contents
0	The modification set(s) used to produce the present COSPL from the last released COSPL, the job that generated the present COSPL, UPDATE output from the generating job, a job for creating a local version of COS, Eclipse procedure files for COSPL, and a report describing all modifications made by this release and all Eclipse procedure files used.
1	All the binaries generated from COSPL plus $SYSTXT and COSTXT. It also contains startup parameter file and station command files for installing the system.
2	The present COSPL
3	A copy of file 0
4	A copy of file 1
5	A copy of file 2

COSPL Listings

This tape contains source listings of the decks in COSPL. To print the listing on file n, mount the tape and type OUT MTO:n.

File	Contents
0	$SYSTXT listing
1	COSTXT listing
2	EXEC listing
3	CSP listing
4	DDC listing
5	TAB listing
6	COM listing
7	CIO:TIO listing
8	STP:Z listing

File	Contents
9	SCP listing
10	EXP listing
11	PDM listing
12	DQM listing
13	MSG listing
14	MEP listing
15	JSH listing
16	DEC listing
17	AUDIT listing
18	COMPARE listing
19	COPY listing
20	DEBUG listing
21	DSDUMP listing
22	DUMP listing
23	EXTRACT listing
24	PDSDUMP listing
25	PDSLOAD listing
26	SKIP listing
27	STATS listing
28	UNB listing
29	WRITEDS listing

PRODPL

This tape contains all of the software associated with the PRODPL program library.

File	Contents
0	The modification set(s) used to produce the present PRODPL from the last released PRODPL, UPDATE output(s) from the PRODPL generation, and Eclipse procedure files for PRODPL.
1	All the binaries generated from PRODPL
2	The present PRODPL
3	A copy of file 0
4	A copy of file 1
5	A copy of file 2

PRODPL Listings

This tape contains source listings of the decks in PRODPL. To print the listing on file n, mount the tape and type OUT MTO:n.

File	Contents
0	CAL listing
1	LDR listing
2	UPDATE listing
3	BUILD listing
4	CSIM listing
5	FDUMP listing

CFTPL with Listings

This tape contains all of the software associated with the CFTPL program library and a source listing of CFT. To print the listing, mount the tape and type OUT MTØ:6

File	Contents
0	Modification set(s) used to produce the present CFTPL from the last released CFTPL, output from the CFTPL generation, and Eclipse procedure files for CFTPL.
1	CFT binary
2	Present CFTPL
3	A copy of file 0
4	A copy of file 1
5	A copy of file 2
6	CFT listing

FTLIBPL with Listings

This tape contains all of the software associated with the FTLIBPL, SCILBPL, and SYSLBPL program libraries and source listings of all the decks within FTLIBPL, SCILBPL, and SYSLBPL. To print the complete FTLIBPL listing, mount the tape and type OUT MTØ:6. To print the complete SCILBPL listing, mount the tape and type OUT MTØ:7. To print the complete SYSLBPL listing, mount the tape and type OUT MTØ:8.

File	Contents
0	Modification set(s) used to produce the present FTLIBPL from the last released FTLIBPL, modification set(s) used to produce the present SCILBPL from the last released SCILBPL, modification set(s) used to produce the present SYSLBPL from the last released SYSLBPL, and Eclipse procedure files for FTLIBPL, SCILBPL, and SYSLBPL.
1	$FTLIB, $SCILIB, and $SYSLIB binary libraries
2	Present FTLIBPL, SCILBPL, and SYSLBPL
3	A copy of file 0
4	A copy of file 1
5	A copy of file 2
6	$FTLIB listing
7	$SCILIB listing
8	$SYSLIB listing

DGS software

Two tapes contain all of the software associated with the Eclipse station.

The Eclipse software tape contains the following files:

File	Contents
0	Procedure files
1	Utilities and command files (i.e., a copy of deadstart tape file 9)
2	STATPL
3	Load maps
4	Library files, SLA library files, binary files, symbol tables
5	Copy of file 0
6	Copy of file 1
7	Copy of file 3
8	Copy of file 4

The Eclipse software listings tape contains the following files:

File	Contents
0	Utilities
1	Local station
2	SLA software
3	Concentrator
4	Remote station

1.2 INITIALIZATION PROCEDURE

This procedure describes the steps necessary for creating an Install disk pack and an Operations disk pack and installing the CRAY-OS Operating System. The Install pack is used for installing the released or customized system; the Operations pack is used when the ECLIPSE is used as a job entry station or operator station and can be used for deadstart or restart but not for install.

This procedure assumes the reader is familiar with RDOS and DGS operation.

1. Initialize the packs.
   a. Label one ECLIPSE pack as "CRI INSTALL" and another as "CRI OPERATIONS" with visual identifiers.
   b. Mount the Install pack and the latest RDOS tape and initialize the pack using the procedure described in the DGS Operator's Guide, CRI publication 2240006.
   c. Repeat step 1b for the Operations pack.
2. Copy Cray Research software onto the Install pack, as follows:
   a. Mount the Install pack.
   b. Mount each of the four PL tapes in turn, copying files 1 and 2 of each tape onto the Install pack. When this step is completed, all of the Cray Research software is on the Install pack.
3. If there are any disk flaws on the master device, modify the INSTALL parameter file to flaw those parts of the disk. (See part 3 for parameter file options and formats.)
4. Install the system on the CRAY-1.
   a. On the 1440, type DELETE $STAT-.- to delete any existing input and output queue files and any print output.
   b. On the 1440, type STATION to bring up the station.
   c. On the 455, type @INSTALL.CM. This initiates a system startup with the install option, stages in all system programs, and builds the system directory.
   d. When the station display shows that there are 0 input files left and there is 1 output file, the system should be installed. Verify that the system is running, type OUTPUT on the 1440. This prints the logfile of SYSDIR, the job that builds the system directory.
   e. Type @QUEDS.CM on the 455. This command file stages in the six CRI UPDATE program libraries and any other datasets that the local

site wants made permanent at install time. The command file can be updated as more datasets are added to the Install pack. The staging is complete when the station display again shows 0 input files.

The basic system has been installed on the CRAY-1 and the CRAY-1 disk storage units at this time.

5. Selectively copy software onto the Operations pack:

   a. Remove the Install pack by typing END on the 455 and then CLEAR/A and RELEASE DPØ on the 1440.

   b. Mount the Operations pack and mount the COSPL tape. Selectively load COS from file 1.

When this step is complete, COS has been initialized and is ready for jobs. Usually, one of the first jobs run is a job that generates a local system that is tailored for the site's hardware configuration and for local software preferences (see section 2).

CAUTION

When working with the released PLs, make temporary modifications local to the job only. Do not dispose these PLs. The released PLs must be the only ones known to the system.

COS MODIFICATION 2

2.1 INSTALLATION DEPENDENT MODIFICATIONS

Two types of installation dependent modifications are described in this section. The first, installation-dependent parameters, describes the basic site hardware configuration, scheduling parameters, and defaults. The second type consists of hardware parameters such as disk flaws.

2.1.1 INSTALLATION PARAMETERS

Installation parameters represent those values in the system that can be adjusted to the needs of the site either by modifying and reassembling the COSTXT or EXEC portions of the system (installation-dependent parameters) or by modifying constants in STP while the system is running (installation-variable parameters). Both types begin with the characters I@. Below is a listing of all installation parameters in alphabetical order. The values are those assembled into the released version of the system.

Installation-dependent parameters set by equates in COSTXT are flagged with CDs, and those set by equates in $SYSTXT are flagged with SDs. Those in EXEC are flagged with EDs. Installation-variable parameters set by CON instructions in STP are flagged with SVs.

Parameter	Type	Value	Significance
I@AGECP	SV	0	Flag enabling the aging of CPU priorities, if set. If this flag is zero, CPU priorities are not aged, and memory priorities are used instead for scheduling the CPU.
I@ALLSDT	SV	1	Flag enabling the display of all SDT entries on all station displays

Parameter	Type	Value	Significance
I@AUTOFL	SV	0	Flag disabling all reductions in user field length (except from RFL statements) if zero
I@BFI	CD,SD	033_8	ASCII character to be used as default Blank Field Initiator (777_8 indicates no blank field compression)
I@BULLIT	CO	0	System bulletin dataset option: 0 No bulletin dataset 1 Bulletin dataset
I@CONFLT	SV	80,000,000*2	Initial time during which the job's field length is not allowed to shrink (two seconds)
I@CPDBT	CD	74,000	Minimum number of cycles needed to transfer one disk block; used in CPU priority calculations.
I@CPMULT	SV	1	Flag making CPU priority depend on memory priority, if set.
I@CPPRI	SV	1	Flag governing CPU priorty calculations. See part 1, section 3.8.2.
I@CSDMAX	CD	512	Maximum job class structure definition size.
I@DCUMC	ED	8	Maximum number of disk master clears during deadstart
I@DEFLM	CD	100,000	Default dataset size limit.
I@DKRTRY	SV	18	Maximum disk retries.
I@DMPSIZ	CD	4005000_8	Size, in words, to reserve for system dump. Two additional sectors will be reserved for dump header, etc.
I@DPWAIT	SD,CD	0	Default value for DISPOSE=WAIT 0 NOWAIT 1 WAIT
I@DSCERR	CD	1	Option for Startup handling of DSC entries containing fatal errors. 0 Delete from DSC 1 Flag and retain DSC entry

Parameter	Type	Value	Significance
I@DSPINC	CD	128	Fixed number of words added to DSP area when the D bit is set in a memory request
I@DTRDLY	CD	4	Dataset transfer postpone delay count
I@DVLRES	CD	2	Number of tracks reserved for writing device labels
I@ENQRT	ED	128	Maximum number of delayed returns queued in EXEC; must be an integer multiple of 64.
I@ERASE	CD	0	Leftmost 16 bits of fill word for user areas
I@FEMSK	CD	-1	Bit mask allowing user control over fatal errors: 0 Error is fatal for job 1 Error is fatal for job step
I@FLINC	SV	0.5	Priority increment for minimum job size; used in computing initial memoty priorities. Allowable range is 0.0 - 15.0.
I@IAPOLL	SV	40,000,000	Delay interval in RTC units between responses to synchronous mode terminals if no output is queued
I@IJTL	CD	7000_8	Initial length of the JTA. This value must be a multiple of 1000_8; 5000_8 is the minimum value.
I@INQLIM	SV	NE@SDT/2	Maximum input queue size
I@JCCHAR	CD	16	Maximum job class characteristic size, in words
I@JFLDEF	SV	145000_8	Default job size, in words, including JTA
I@JFLMAX	SV	3400000_8	Maximum job size, in words, including JTA
I@JFLMIN	SV	22000_8	Minimum initial job size, in words, including JTA
I@JFLMSG	SV	0	Flag disabling field-length change and open/close messages, if 0

Parameter	Type	Value	Significance
I@JOBMAX	SV	4	Maximum number of jobs in memory; allowable range is 0 to I@JXTSIZ.
I@JSHAMP	SV	1.0	Amplitude of priorty curves; used in computing memory priority. Allowable range, 0.0-7.0.
I@JSHDB	SV	0.6	Deadband; used in comparing memory priorities. Allowable range, 0.0-7.0.
I@JSHDR	SV	0.2	Decay rate; used in computing memory priority. Allowable range, 0.0-1.0.
I@JSHJSW	SV	1.0	Weighting factor for job size; used in computing memory priority. Allowable range, 0.0-1.0.
I@JSHPHW	SV	1	Past history (damping) factor; used in computing CPU priority. Allowable range, 1-8. Higher values cause greater damping of changes in CPU priorty.
I@JSHRR	SV	0.05	Rise rate; used in computing memory priorty. Allowable range, 0.0-1.0.
I@JSHSI	SV	80,000,000*5	Memory scheduling interval (five seconds)
I@JSHTLE	SV	80,000,000*3	Amount of time limit extension (three seconds)
I@JTLDEF	CD	8	Default value for the job time limit in seconds
I@JXTSIZ	CD	15	Number of jobs that can run in this system. Allowable range, 1-63.
I@LGBSZ	CD	2000_8	$SYSLOG buffer size in words
I@LGDSZ	CD	2000	$SYSLOG dataset size in sectors
I@LGUSZ	CD	60_8	$LOG size limit in blocks
I@MAXDSZ	SZ	512*411*2	Maximum dataset size
I@MAXLM	CD	20000	Maximum dataset size limit
I@MAXWPT	CD	512*18	Maximum track size in words for all disks

Parameter	Type	Value	Significance
I@MEM	CD	1000000_8	Memory size for minimum hardware configuration
I@MEMPRI	CD	7	Default value when no P parameter is given on JOB statement
I@MINDEC	CD	2000_8	Threshold value for field length decrease. Field length reductions less than this amount are not performed.
I@MINWPT	CD	512*18	Minimum track size, in words, for all disks
I@MP1SZ	CD	2000	Size of memory pool 1
I@MP2SZ	CD	500	Size of memory pool 2
I@MXDRET	ED	3	Maximum number of disk retries in EXEC DD-19/DD-29 Disk Driver
I@NCLASS	CD	16	Maximum number of job classes that can be defined
I@NORRN	CD	0	Enable no rerun checking
I@NPD	CD	1000	Maximum number of permanent datasets
I@PDRT	CD	1	Default permanent dataset retention period in days
I@PDSBFL	CD	512*18	PDSDUMP/PDSLOAD buffer size; must be a multiple of 512.
I@PRLVL	CD	8	Maximum procedure level, including primary level
I@PRSZW	CD	60_8	Weighting factor (given as a percentage) used in determining output queue priority based on dataset size
I@RFLMIM	SV	6000_8	Minimum job size after performing a reduction in field length
I@RRJ	CD	1	Default if recovery of rolled jobs enabled for a Restart
I@RRN	CD	0	Enable job rerun

Parameter	Type	Value	Significance
I@SDR	CD	0	Flag enabling SDR recovery 0 Recover SDR 1 Do not recover SDR
I@SFLMIN	SV	13000_8	Maximum initial job size if on simulator
I@SIMBFZ	SV	1	Default buffer size if on simulator
I@SIMDSZ	CD	400000_8	Size in words to reserve for system dump when executing in simulator
I@SPMDLY	SV	1800	SPM collection interval in seconds
I@SPMMIN	SV	10	Delay when SPM needs memory (seconds)
I@SPMON	SV	1	SPM task enable flag
I@SPMTYP	SV	3454_8	SPM subtype enable vector
I@STRTHR	SV	264_8	Block transfer limit for streaming
I@TLBIAS	SV	16	Time limit bias (chosen empirically); used for computing initial memory priorities.
I@TSCTM	SV	2.0	Connect time multiplier; used for computing variable time slice. Allowable range, 0.0-10.0.
I@TSMIN	SV	0	Minimum time slice in milliseconds. If I@TSMIN is 0, the maximum value allowed (32767) is used.
I@TSMPM	SV	8.0	Memory priority multiplier used when computing a variable time slice; allowable range, 0.0-50.0.
I@ZOPT	CD	3	System startup default option: 1 Install 2 Restart 3 Deadstart
I@$INS2	SV	4	Default CIO buffer size for job's $IN dataset
I@$ODLM	CD	2000	Default $OUT size limit
I@$OMLM	CD	20000_8	Maximum $OUT size limit

I@$OUTSZ	SV	4	Default CIO buffer size for job's $OUT dataset
I@819MS	SV	18	Number of micro positions for 819

2.1.2 HARDWARE PARAMETERS

Hardware parameters represent those values in the system that depend wholly on the number and type of peripherals present at the site. These parameters provide the site with the ability to specify significant characteristics of peripherals.

Disk flaws

Disk flaws and tracks reserved for engineering diagnostics can be reserved during system assembly of STP through the FLAW macro (part 3, section 1). There are no default values for such parameters. The FLAW macro is contained in COSTXT and uses the helper macros STRTFLW, ENDFLW, and %GETNUM. The sequence of calls needed to reserve tracks is:

```
STRTFLW  drtname, DT=disk-type
FLAW     flaw1
  :
FLAW     flawn
ENDFLW
```

The STRTFLW macro indicates which device is being flawed. The *drtname* parameter must be specified and is the CAL symbol that identifies word 0 of the DRT header. In the released system, these are DRT00, DRT01,...,DRT17. The *disk-type* parameter can be DD19 or DD29 and must be specified.

NOTE

The *drtname* is *not* the logical device name. Also, be sure that the DRT definition occurs before the STRTFLW macro is called because the SET pseudo instruction references *drtname*, and the assembler requires that symbols used on SET pseudo instructions be previously defined.

The ENDFLW macro terminates the setting of flaws for a device. It must be present to prevent subsequent code from being assembled into the DRT bit map. ENDFLW uses an ORG pseudo instruction to position to the word address having the value of the origin counter prior to the most recent STRTFLW macro.

The FLAW macro specifies a track or range of tracks to be reserved. A single track, a range of tracks within one cylinder, or a range of entire cylinders may be specified but sector numbers currently are treated as documentation only and the entire specified track is flawed. A FLAW macro call has one of the following formats (in which all numbers are octal):

FLAW	C*nnn*	Flaw all tracks of cylinder *nnn*.
FLAW	C*nnn*-*mmm*	Flaw all tracks of all cylinders between *nnn* and *mmm*, inclusively.
FLAW	C*nnn*,T*mm*	Flaw track *mm* of cylinder *nnn*.
FLAW	C*nnn*,T*mm*-*pp*	Flaw tracks *mm* through *pp*, inclusively, on cylinder *nnn*.
FLAW	C*nnn*,T*mm*,S*ss*	Same as FLAW C*nnn*,T*mm*
FLAW	C*nnn*,T*mm*,S*ss*-*tt*	Same as FLAW C*nnn*,T*mm*

Setting the DRT header to reflect the number of allocation units flawed is not necessary. Routine Z of Startup initializes the count correctly without using the DRAIA value assembled into STP.

Configuring additional station channels

The default system has only an MCU station configured. To add other station channels, the installation must expand the Link Configuration Table (LCT). The LCT allows separate configuration of each channel.

To configure an additional station channel, the installation must increase the number of active physical channels (field LCNA) in the LCT header and add an LCT entry for the new station channel. The LCT entry includes:

- The channel pair number (1-12).
- The maximum number of logical IDs to be supported by the channel.

- The number of streams per physical channel. The total number of streams is the sum of the maximum number of streams for each logical ID.
- The size of the disk buffer in words; this must be a multiple of 512. In general, large disk buffers will increase throughout at the sacrifice of memory.
- The maximum allowable segment size. The segment buffer size must be set to the size of the largest segment to be used.

Changing disk unit configuration

The hardware configuration of disk units is communicated to COS through the following tables:

DCT	Device Channel Table
PUT	Physical Unit Table
EQT	Equipment Table
DRT	Disk Reservation Table plus associated FLAW macros
AET	Attached Equipment Table

The DCT, EQT, and DRT are located in STP, while the PUT and AET are in EXEC. The Disk Queue Manager (DQM) moves information from the DCT to the disk driver. The DCT, which contains an entry for each disk channel, is set up initially to manage 16 disk units (four per channel) and need not be changed unless more disk channels are present. Each entry in the DCT initially contains pointers to the first EQT assigned to that channel and to the number of units on that channel.

The EQT, PUT, and DRT are closely related. Each table has an entry for each disk unit in the system.

The order of entries in the EQT determines the selection sequence for the assignment of new datasets. Therefore, all of the first units on each channel must appear first, all of the second units next, and so on. The PUT and DRT entries must appear unit for unit in the same order. Each EQT entry points to the next EQT on the same channel. The last EQT of each channel is linked back to the first, forming a circular linked list.

The following table entries represent a configuration of 16 disk units (four units per channel). Only one EQT entry can be defined as the master device (MSD=1).

EQUIPMENT TABLE (EQT)

```
EQT00    EQT    LDV='DD-19-20',CH1=2,UT1=0,DRT=DRT00,DT=DD19,NA=0,OFF=0,MSD=1,LNK=EQT04
EQT01    EQT    LDV='DD-19-30',CH1=3,UT1=0,DRT=DRT01,DT=DD19,NA=1,OFF=1,LNK=EQT05
EQT02    EQT    LDV='DD-19-40',CH1=4,UT1=0,DRT=DRT02,DT=DD19,NA=1,OFF=1,LNK=EQT06
EQT03    EQT    LDV='DD-19-50',CH1=5,UT1=0,DRT=DRT03,DT=DD19,NA=1,OFF=1,LNK=EQT07
EQT04    EQT    LDV='DD-19-21',CH1=2,UT1=1,DRT=DRT04,DT=DD19,NA=1,OFF=1,LNK=EQT10
EQT05    EQT    LDV='DD-19-31',CH1=3,UT1=1,DRT=DRT05,DT=DD19,NA=1,OFF=1,LNK=EQT11
EQT06    EQT    LDV='DD-19-41',CH1=4,UT1=1,DRT=DRT06,DT=DD19,NA=1,OFF=1,LNK=EQT12
EQT07    EQT    LDV='DD-19-51',CH1=5,UT1=1,DRT=DRT07,DT=DD19,NA=1,OFF=1,LNK=EQT13
EQT10    EQT    LDV='DD-19-22',CH1=2,UT1=2,DRT=DRT10,DT=DD19,NA=1,OFF=1,LNK=EQT14
EQT11    EQT    LDV='DD-19-32',CH1=3,UT1=2,DRT=DRT11,DT=DD19,NA=1,OFF=1,LNK=EQT15
EQT12    EQT    LDV='DD-19-42',CH1=4,UT1=2,DRT=DRT12,DT=DD19,NA=1,OFF=1,LNK=EQT16
EQT13    EQT    LDV='DD-19-52',CH1=5,UT1=2,DRT=DRT13,DT=DD19,NA=1,OFF=1,LNK=EQT17
EQT14    EQT    LDV='DD-19-23',CH1=2,UT1=3,DRT=DRT14,DT=DD19,NA=1,OFF=1,LNK=EQT00
EQT15    EQT    LDV='DD-19-33',CH1=3,UT1=3,DRT=DRT15,DT=DD19,NA=1,OFF=1,LNK=EQT01
EQT16    EQT    LDV='DD-19-43',CH1=4,UT1=3,DRT=DRT16,DT=DD19,NA=1,OFF=1,LNK=EQT02
EQT17    EQT    LDV='DD-19-53',CH1=5,UT1=3,DRT=DRT17,DT=DD19,NA=1,OFF=1,LNK=EQT03
```

DISK RESERVATION TABLE (DRT)

```
DRT00    DRT    LDV='DD-19-20',DT=DD19
DRT01    DRT    LDV='DD-19-30',DT=DD19
DRT02    DRT    LDV='DD-19-40',DT=DD19
DRT03    DRT    LDV='DD-19-50',DT=DD19
DRT04    DRT    LDV='DD-19-21',DT=DD19
DRT05    DRT    LDV='DD-19-31',DT=DD19
DRT06    DRT    LDV='DD-19-41',DT=DD19
DRT07    DRT    LDV='DD-19-51',DT=DD19
DRT10    DRT    LDV='DD-19-22',DT=DD19
DRT11    DRT    LDV='DD-19-32',DT=DD19
DRT12    DRT    LDV='DD-19-42',DT=DD19
DRT13    DRT    LDV='DD-19-52',DT=DD19
DRT14    DRT    LDV='DD-19-23',DT=DD19
DRT15    DRT    LDV='DD-19-33',DT=DD19
DRT16    DRT    LDV='DD-19-43',DT=DD19
DRT17    DRT    LDV='DD-19-53',DT=DD19
```

PHYSICAL UNIT TABLE (PUT)

```
U20      PUT    ICH=4,LUT=0,DT=DD19
U21      PUT    ICH=4,LUT=1,DT=DD19
U22      PUT    ICH=4,LUT=2,DT=DD19
U23      PUT    ICH=4,LUT=3,DT=DD19
U30      PUT    ICH=6,LUT=0,DT=DD19
U31      PUT    ICH=6,LUT=1,DT=DD19
U32      PUT    ICH=6,LUT=2,DT=DD19
U33      PUT    ICH=6,LUT=3,DT=DD19
U40      PUT    ICH=10,LUT=0,DT=DD19
U41      PUT    ICH=10,LUT=1,DT=DD19
U42      PUT    ICH=10,LUT=2,DT=DD19
U43      PUT    ICH=10,LUT=3,DT=DD19
U50      PUT    ICH=12,LUT=0,DT=DD19
U51      PUT    ICH=12,LUT=1,DT=DD19
U52      PUT    ICH=12,LUT=2,DT=DD19
U53      PUT    ICH=12,LUT=3,DT=DD19
```

The tables demonstrate the use of DRT, EQT, and PUT macros supplied in COSTXT. Keywords used in these macros are defined as follows:

LDV	Logical device name
CH1	Primary channel number
UT1	Primary unit number
DRT	DRT address
DT	Disk type, either DD19 or DD29
NA	Unit not available
OFF	Unit off
MSD	Master device flag
LNK	Link to next equipment on channel
ICH	Input channel number (twice the value of CH1)
LUT	Logical unit number

2.2 LOCAL SYSTEM GENERATION

This procedure assumes that as a minimum, the COS initialization procedure described in section 1.2 has been completed. The Operations pack should be mounted.

1. Write the local modifications in UPDATE format, block them, and stage them in as a dataset named LOCIN.
2. Load LOCJOB from file 0 of the COSPL tape and run LOCJOB. This job disposes to the station, COSLOC (local system binary), DDCLOC (local DDC binary), ESYMLOC (EXEC symbol table), SSYMLOC (STP symbol table) and SLSLOC (STP listing).
3. By convention, COS is the name linked to the production system and DDC, SSYM, and ESYM are the names linked to the corresponding dead dump creator and symbol tables for FDUMP processing. Rename and relink as follows:
   a. Rename COSLOC, DDCLOC, ESYMLOC and SSYMLOC to COSmmdd, DDCmmdd, ESYMmmdd and SSYMmmdd, respectively, where mmdd is the month and day of generation.
   b. Link COS, DDC, ESYM and SSYM to COSmmdd, DDCmmdd, ESYMmmdd, SSYMmmdd respectively. If any name already exists as a link, unlink it. If any name already exists as a file, rename it as in step a. Note the DDC must exist in DPØ in order for the SYSDUMP command to work.
   c. Stage ESYM and SSYM to the CRAY for use by FDUMP.
   d. A system start-up of COS will bring the newly generated local system into operation.
   e. Save COSmmdd, DDCmmdd, ESYMmmdd and SSYMmmdd on tape and load onto the INSTALL pack. Follow the same naming and linking procedure used in step b.

SAMPLE UPDATE JOBS 3

3.1 COSGEN JOB

The following is a sample statusing job used for creating the entire released system. The user may consider this job as a guideline for designing jobs to modify decks in COSPL. However, in no case should new program libraries be disposed to the station.

```
JOB,JN=COSGEN,M=600,T=2000.
*.
*.        GENERATE CRAY-1 OPERATING SYSTEM FOR RELEASE.
*.
*.
*.        GENERATE FTLIBNL FROM FTLIBPL AND FTLIBIN
*.
ACCESS,DN=FTLIBPL.
ACCESS,DN=FTLIBIN.
UPDATE,P=FTLIBPL,I=FTLIBIN,N=FTLIBNL,C=0,L=LIBOUT,ED,ID.
DISPOSE,DN=LIBOUT,DC=ST,MF=LS.
RELEASE,DN=FTLIBPL.
RELEASE,DN=FTLIBIN.
*.
*.        GENERATE SYSLBNL FROM SYSLBPL AND SYSLBIN
*.
ACCESS,DN=SYSLBPL.
ACCESS,DN=SYSLBIN.
UPDATE,P=SYSLBPL,I=SYSLBIN,N=SYSLBNL,C=0,L=SYSOUT,ED,ID.
DISPOSE,DN=SYSOUT,DC=ST,MF=LS.
RELEASE,DN=SYSLBPL.
RELEASE,DN=SYSLBIN.
*.
*.
*.        GENERATE SCILBNL FROM SCILBPL AND SCILBIN
*.
ACCESS,DN=SCILBPL.
ACCESS,DN=SCILBIN.
UPDATE,P=SCILBPL,I=SCILBIN,N=SCILBNL,C=0,L=SCIOUT,ED,ID.
DISPOSE,DN=SCIOUT,DC=ST,MF=LS.
RELEASE,DN=SCILBPL.
RELEASE,DN=SCILBIN.
*.
*.        GENERATE PRODNL FROM PRODPL AND PRODIN
*.
ACCESS,DN=PRODPL.
ACCESS,DN=PRODIN.
UPDATE,P=PRODPL,I=PRODIN,N=PRODNL,C=0,L=PRODOUT,ED,ID,DW=80.
DISPOSE,DN=PRODOUT,DC=ST,MF=PS.
RELEASE,DN=PRODPL.
RELEASE,DN=PRODIN.
```

Sample COSGEN job (continued)

```
*.
*.        GENERATE CFTNL FROM CFTPL AND CFTIN
*.
ACCESS,DN=CFTPL.
ACCESS,DN=CFTIN.
UPDATE,P=CFTPL,I=CFTIN,N=CFTNL,C=0,L=CFTOUT,ED,ID,*=%
DISPOSE,DN=CFTOUT,DC=ST,MF=FS.
RELEASE,DN=CFTPL.
RELEASE,DN=CFTIN.
*.
*.        GENERATE COSNL FROM COSPL AND COSIN
*.
ACCESS,DN=COSPL.
ACCESS,DN=COSIN.
UPDATE,P=COSPL,I=COSIN,N=COSNL,C=0,L=COSOUT,ED,ID.
DISPOSE,DN=COSOUT,DC=ST,MF=CS.
RELEASE,DN=COSPL.
RELEASE,DN=COSIN.
*.
*.
*.        $SYSTXT,COSTXT,SKOLTXT,$FTLIB,$SYSLIB,SCILIB ARE PARTS OF THE
*.        INPUT TO OTHER BINARIES SO THEY ARE NOT DISPOSED UNTIL THE END
*.        OF THE JOB.  CAL,LDR,UPDATE,BUILD,CFT,UNB,SKOL ARE USED TO GENERATE
*.        BINARIES SO THEY ARE NOT DISPOSED UNTIL THE END OF THE JOB.
*.
*.        GENERATE $SYSTXT
*.
UPDATE,P=COSNL,C=$SYSTXT,Q=ST,I=0.
CAL,I=$SYSTXT,B=0,S=0,L=STLS,ABORT,E.
DISPOSE,DN=STLS,DC=ST,MF=CS.
*.
*.        GENERATE COSTXT
*.
UPDATE,P=COSNL,C=COSTXT,Q=CT,I=0.
CAL,I=COSTXT,B=0,S=$SYSTXT,L=CTLS,ABORT,E.
DISPOSE,DN=CTLS,DC=ST,MF=CS.
*.
*.        GENERATE SKOL
*.
*.        BECAUSE SKOL IS NEW PRODUCT WITH THE 1.08 RELEASE AND IS USED
*.        IN GENERATING 1.08 $FTLIB.  SKOL MUST BE GENERATED BEFORE $FTLIB.
*.
UPDATE,P=PRODNL,C=CF,Q=SKOL,I=0.
CFT,I=CF,L=SKOLLS,OFF=BCT.
LDR,AB=SKOL,MAP=FULL,NX,L=SKOLLS.
REWIND,DN=SKOL.
REWIND,DN=CF.
DISPOSE,DN=SKOLLS,DC=ST,MF=PS.
REWIND,DN=$BLD.
*.
*.       GENERATE SKOLTXT
*.
UPDATE,P=PRODNL,C=SKOLTXT,Q=SKOLTXT,I=0,DW=80.
*.
*        TO PRINT A LISTING OF SKOLTXT, USE 'OUT/S SKOLTXT'
```

Sample COSGEN job (continued)

```
*.
*.        GENERATE $FTLIB
*.
UPDATE,P=FTLIBNL,I=0,C=CF,F.
CAL,I=CF,L=LIBLS,ABORT,E.
CFT,I=CF,L=LIBLS,OFF=BCT.
ASSIGN,DN=CF,A=FT17.
SKOL,I=FT17,L=LIBLS.
CFT;I=$SKF,L=0,E=5.
LDR,MAP=FULL,L=LIBLS,NX.
RELEASE,DN=$FTLIB.
BUILD,I=0,OBL=0,NBL=$FTLIB,L=LIBLS.
REWIND,DN=$FTLIB.
DISPOSE,DN=LIBLS,DC=ST,MF=LP.
RELEASE,DN=CFL.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE $SYSLIB
*.
UPDATE,P=SYSLBNL,I=0,C=CF,F.
CAL,I=CF,L=SYSLS,S=$SYSTXT:COSTXT,ABORT,E.
CFT,I=CF,L=SYSLS,OFF=BCT.
LDR,MAP=FULL,L=SYSLS,NX.
RELEASE,DN=$SYSLIB.
BUILD,I=0,OBL=0,NBL=$SYSLIB,L=SYSLS.
REWIND,DN=$SYSLIB.
DISPOSE,DN=SYSLS,DC=ST,MF=LP.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE $SCILIB
*.
UPDATE,P=SCILBNL,C=CF,Q.
CAL,I=CF,L=SCILS,ABORT,E.
REWIND,DN=CF.
UPDATE,P=SCILBNL,C=CF,Q.
CFT,I=CF,L=SCILS,OFF=BCT.
LDR,MAP=FULL,L=SCILS,NX.
RELEASE,DN=$SCILIB.
BUILD,I=0,OBL=0,NBL=$SCILIB,L=SCILS.
DISPOSE,DN=SCILS,DC=ST,MF=LP.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE UPDATE
*.
UPDATE,P=PRODNL,C=CF,Q=UF,I=0.
CFT,I=CF,L=UDLS,OFF=BCT.
REWIND,DN=CF.
UPDATE,P=PRODNL,C=CF,Q=UC,I=0.
CAL,I=CF,L=UDLS,ABORT,E.
LDR,AB=UPDATE,MAP=FULL,L=UDLS,NX.
REWIND,DN=UPDATE.
DISPOSE,DN=UDLS,DC=ST,MF=PS.
REWIND,DN=CF.
REWIND,DN=$BLD.
```

Sample COSGEN job (continued)

```
*.
*.        GENERATE CAL
*.
UPDATE,P=PRODNL,C=CF,Q=C,I=0.
CAL,I=CF,L=CLS,ABORT,E.
LDR,AB=CAL,MAP=FULL,L=CLS,NX.
REWIND,DN=CAL.
DISPOSE,DN=CLS,DC=ST,MF=PS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE CFT
*.
UPDATE,P=CFTNL,C=CF,Q=F,I=0,*=%.
CAL,I=CF,L=CFTLS,ABORT,E.
LDR,AB=CFT,NX.
REWIND,DN=CFT.
DISPOSE,DN=CFTLS,DC=ST,MF=FS.
REWIND,DN=$BLD.
REWIND,DN=CF.
*.
*.        GENERATE LDR
*.
UPDATE,P=PRODNL,C=CF,Q=L,I=0.
CAL,I=CF,L=LLS,ABORT,E.
LDR,AB=LDR,MAP=FULL,L=LLS,NX.
REWIND,DN=LDR.
DISPOSE,DN=LLS,DC=ST,MF=PS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE BUILD
*.
UPDATE,P=PRODNL,C=CF,Q=BF,I=0.
CFT,I=CF,L=BLS,OFF=BCT.
LDR,AB=BUILD,MAP=FULL,L=BLS,NX.
REWIND,DN=BUILD.
DISPOSE,DN=BLS,DC=ST,MF=PS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE UNB
*.
UPDATE,P=COSNL,C=CF,Q=UN,I=0.
CAL,I=CF,L=UNLS,ABORT,E.
LDR,AB=UNB,MAP=FULL,L=UNLS,NX.
REWIND,DN=UNB.
DISPOSE,DN=UNLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        THE FOLLOWING ARE IN ALPHABETICAL ORDER
*.
*.        GENERATE AUDIT
*.
UPDATE,P=COSNL,C=CF,Q=AF,I=0.
CFT,I=CF,L=AULS,OFF=BCT.
```

Sample COSGEN job (continued)

```
REWIND,DN=CF.
UPDATE,P=COSNL,C=CF,Q=AC,I=0.
CAL,I=CF,L=AULS,ABORT,E.
LDR,AB=AUDIT,MAP=FULL,L=AULS,NX.
REWIND,DN=AUDIT.
DISPOSE,DN=AUDIT,DC=ST,MF=CP.
DISPOSE,DN=AULS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.          GENERATE COMPARE
*.
UPDATE,P=COSNL,C=CF,Q=CM,I=0.
CFT,I=CF,L=CMLS,OFF=BCT.
REWIND,DN=CF.
UPDATE,P=COSNL,C=CF,Q=CMC,I=0.
CAL,I=CF,L=CMLS,ABORT,E.
LDR,AB=COMPARE,MAP=FULL,L=CMLS,NX.
REWIND,DN=COMPARE.
DISPOSE,DN=COMPARE,DC=ST,MF=CP.
DISPOSE,DN=CMLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.          GENERATE COPYD
*.
UPDATE,P=COSNL,C=CF,Q=CD,I=0.
CFT,I=CF,L=CPLS,OFF=BTC.
LDR,AB=COPYD,MAP=FULL,L=CPLS,NX.
REWIND,DN=COPYD.
DISPOSE,DN=COPYD,DC=ST,MF=CP.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.          GENERATE COPYF
*.
UPDATE,P=COSNL,C=CF,Q=CF,I=0.
CFT,I=CF,L=CPLS,OFF=BTC.
LDR,AB=COPYF,MAP=FULL,L=CPLS,NX.
REWIND,DN=COPYF.
DISPOSE,DN=COPYF,DC=ST,MF=CP.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.          GENERATE COPYR
*.
UPDATE,P=COSNL,C=CF,Q=CR,I=0.
CFT,I=CF,L=CPLS,OFF=BTC.
LDR,AB=COPYR,MAP=FULL,L=CPLS,NX.
REWIND,DN=COPYR.
DISPOSE,DN=COPYR,DC=ST,MF=CP.
DISPOSE,DN=CPLS,DC=ST,MF=CS.
REWIND,DN=CF.
*.
*.          GENERATE CSIM
*.
UPDATE,P=PRODNL,Q=CSF,C=CF,I=0.
```

Sample COSGEN job (continued)

```
UPDATE,P=PRODNL,C=CSIMTXT,Q=CSIMTXT,I=0.
UPDATE,P=PRODNL,C=CSIMMAC,Q=CSIMMAC,I=0.
SKOL,I=CF,L=CSLS.
CFT,I=$SKF,L=0,B=CSIMR.
RELEASE,DN=CF.
UPDATE,P=PRODNL,Q=CSC,C=CF,I=0.
CAL,I=CF,L=CSLS,B=CSIMR,ABORT,E,ON,LIST,LIS.
LDR,DN=CSIMR,NX,L=CSLS,MAP=FULL,AB=CSIM.
REWIND,DN=CF.
REWIND,DN=CSIMR.
REWIND,DN=CSIMTXT.
REWIND,DN=CSIMMAC.
REWIND,DN=$BLD.
DISPOSE,DN=CSIMR,DC=ST,MF=PP.
DISPOSE,DN=CSIMTXT,DC=ST,MF=PP.
DISPOSE,DN=CSIMMAC,DC=ST,MF=PP.
DISPOSE,DN=CSLS,DC=ST,MF=PS.
RELEASE,DN=CSIM.
*.
*.        GENERATE DDC
*.
UPDATE,P=COSNL,C=CF,Q=DDC,I=0.
CAL,I=CF,L=DDCLS,E,ABORT,S=COSTXT.
REWIND,DN=$BLD.
UNB,I=$BLD,O=DDC.
DISPOSE,DN=DDC,DC=ST,MF=CP.
DISPOSE,DN=DDCLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE DEBUG
*.
UPDATE,P=COSNL,C=CF,Q=DBF,I=0.
CFT,I=CF,L=DBLS,OFF=BCT.
REWIND,DN=DEBUG.
LDR,AB=DEBUG,MAP=FULL,L=DBLS,NX.
REWIND,DN=DEBUG.
DISPOSE,DN=DEBUG,DC=ST,MF=CP.
DISPOSE,DN=DBLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE DSDUMP
*.
UPDATE,P=COSNL,C=CF,Q=DD,I=0.
CAL,I=CF,L=DDLS,ABORT,E.
LDR,AB=DSDUMP,MAP=FULL,L=DDLS,NX.
REWIND,DN=DSDUMP.
DISPOSE,DN=DSDUMP,DC=ST,MF=CP.
DISPOSE,DN=DDLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE DUMP
*.
```

Sample COSGEN job (continued)

```
LDR,AB=DUMP,MAP=FULL,L=DMLS,NX.
REWIND,DN=DUMP.
DISPOSE,DN=DUMP,DC=ST,MF=CP.
DISPOSE,DN=DMLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE EXTRACT
*.
UPDATE,P=COSNL,C=CF,Q=EXF,I=0.
CFT,I=CF,L=EXLS,OFF=BTC.
REWIND,DN=CF.
UPDATE,P=COSNL,C=CF,Q=EXC,I=0.
CAL,I=CF,L=EXLS,ABORT,E.
LDR,AB=EXTRACT,MAP=FULL,L=EXLS,NX.
REWIND,DN=EXTRACT.
DISPOSE,DN=EXTRACT,DC=ST,MF=CP.
DISPOSE,DN=EXLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE FDUMP
*.
UPDATE,P=PRODNL,C=CF,Q=FDF,I=0.
CFT,I=CF,L=FDLS,OFF=BCT.
REWIND,DN=CF.
UPDATE,P=PRODNL,C=CF,Q=FDC,I=0.
CAL,I=CF,L=FDLS,ABORT,E.
LDR,AB=FDUMP,MAP=FULL,L=FDLS,NX.
REWIND,DN=FDUMP.
DISPOSE,DN=FDUMP,DC=ST,MF=PP.
DISPOSE,DN=FDLS,DC=ST,MF=PS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE JCSDEF
*.
UPDATE,P=COSNL,Q=JCF,C=CF,I=0.
CFT,I=CF,L=JCLS,OFF=BCT.
REWIND,DN=CF.
UPDATE,P=COSNL,Q=JCC,C=CF,I=0.
CAL,I=CF,L=JCLS,ABORT,E,S=COSTXT:$SYSTXT.
LDR,AB=JCSDEF,MAP=FULL,L=JCLS,NX.
REWIND,DN=JCSDEF
DISPOSE,DN=JCSDEF,DC=ST,MF=CP.
DISPOSE,DN=JCLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE PDSDUMP
*.
UPDATE,P=COSNL,C=CF,Q=PD,I=0.
CAL,I=CF,S=COSTXT:$SYSTXT,L=PDLS,ABORT,E.
LDR,AB=PDSDUMP,MAP=FULL,L=PDLS,NX.
REWIND,DN=PDSDUMP.
DISPOSE,DN=PDSDUMP,DC=ST,MF=CP.
DISPOSE,DN=PDLS,DC=ST,MF=CS
REWIND DN=CF
```

Sample COSGEN job (continued)

```
REWIND,DN=$BLD.
*.
*.        GENERATE PDSLOAD
*.
UPDATE,P=COSNL,C=CF,Q=PL,I=0.
CAL,I=CF,S=COSTXT:$SYSTXT,L=PLLS,ABORT,E.
LDR,AB=PDSLOAD,MAP=FULL,L=PLLS,NX.
REWIND,DN=PDSLOAD.
DISPOSE,DN=PDSLOAD,DC=ST,MF=CP.
DISPOSE,DN=PLLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE SKIPD, SKIPF, SKIPR
*.
UPDATE,P=COSNL,C=CF,Q=SK,I=0.
CAL,I=CF,L=SKLS,ABORT,E.
LDR,AB=SKIPD,T=SKIPD,MAP=FULL,L=SKLS,NX.
LDR,AB=SKIPF,T=SKIPF,NX.
LDR,AB=SKIPR,T=SKIPR,NX.
REWIND,DN=SKIPD.
DISPOSE,DN=SKIPD,DC=ST,MF=CP.
REWIND,DN=SKIPF.
DISPOSE,DN=SKIPF,DC=ST,MF=CP.
REWIND,DN=SKIPR.
DISPOSE,DN=SKIPR,DC=ST,MF=CP.
DISPOSE,DN=SKLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE STATS
*
UPDATE,P=COSNL,C=CF,Q=STATS,I=0.
CFT,I=CF,L=STATSLS,OFF=BCT.
LDR,AB=STATS,MAP=FULL,L=STATSLS,NX.
REWIND,DN=STATS.
DISPOSE,DN=STATS,DC=ST,MF=CP.
DISPOSE,DN=STATSLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE WRITEDS
*.
UPDATE,P=COSNL,C=CF,Q=WD,I=0.
CAL,I=CF,L=WDLS,ABORT,E.
LDR,AB=WRITEDS,MAP=FULL,L=WDLS,NX.
REWIND,DN=WRITEDS.
DISPOSE,DN=WRITEDS,DC=ST,MF=CP.
DISPOSE,DN=WDLS,DC=ST,MF=CS.
REWIND,DN=CF.
REWIND,DN=$BLD.
*.
*.        GENERATE COS - THIS IS THE LAST STEP IN BINARY GENERATION
*.
UPDATE,P=COSNL,C=E,Q=E,I=0.
UPDATE,P=COSNL,C=S,Q=S,I=0.
UPDATE,P=COSNL,C=J,Q=J,I=0.
```

Sample COSGEN job (continued)

```
CAL,I=E,B=BE,S=COSTXT,L=ELS,LIS,ABORT,E,SYM=EXECSYM.
CAL,I=S,B=BS,S=COSTXT,L=SLS,LIS,ABORT,E,SYM=STPSYM.
CAL,I=J,B=$BLD,S=COSTXT:$SYSTXT,L=JLS,LIS,ABORT,E.
LDR,DN=$BLD,AB=BJ,MAP=FULL,L=JLS,NX.
REWIND,DN=BE.
REWIND,DN=BS.
REWIND,DN=BJ.
REWIND,DN=EXECSYM.
REWIND,DN=STPSYM.
UNB,I=BE,O=COS.
UNB,I=BS,O=COS.
UNB,I=BJ,O=COS.
DISPOSE,DN=COS,DC=ST,MF=CS.
DISPOSE,DN=EXECSYM,DC=ST,MF=CS.
DISPOSE,DN=STPSYM,DC=ST,MF=CS.
DISPOSE,DN=ELS,DC=ST,MF=CS.
DISPOSE,DN=SLS,DC=ST,MF=CS.
DISPOSE,DN=JLS,DC=ST,MF=CS.
*.
*.        GENERATE SEPARATE STP TASK LISTINGS
*.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=TAB,L=TABLS.
DISPOSE,DC=ST,DN=TABLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=COM,L=COMLS.
DISPOSE,DC=ST,DN=COMLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=CIO:TIO,L=TIOLS.
DISPOSE,DC=ST,DN=TIOLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=STP:Z,L=ZLS.
DISPOSE,DC=ST,DN=ZLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=SCP,L=SCPLS.
DISPOSE,DC=ST,DN=SCPLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=EXP,L=EXPLS.
DISPOSE,DC=ST,DN=EXPLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=PDM,L=PDMLS.
DISPOSE,DC=ST,DN=PDMLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=DQM,L=DQMLS.
DISPOSE,DC=ST,DN=DQMLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=MSG,L=MSGLS.
DISPOSE,DC=ST,DN=MSGLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=MEP,L=MEPLS.
DISPOSE,DC=ST,DN=MEPLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=SPM,L=SPMLS.
DISPOSE,DC=ST,DN=SPMLS,MF=CS.
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=JSH,L=JSHLS.
DISPOSE,DC=ST,DN=JSHLS,MF=CS.
```

Sample COSGEN job (continued)

```
REWIND,DN=S.
CAL,I=S,S=COSTXT,B=0,NXNS,E,LIS,LIST=DEC,L=DECLS.
DISPOSE,DC=ST,DN=DECLS,MF=CS.
*.
*.        DISPOSE BINARIES NOT ALREADY DISPOSED
*.
DISPOSE,DN=$SYSTXT,DC=ST,MF=CP.
DISPOSE,DN=COSTXT,DC=ST,MF=CP.
DISPOSE,DN=SKOLTXT,DC=ST,MF=PP.
DISPOSE,DN=$FTLIB,DC=ST,MF=LP.
DISPOSE,DN=$SYSLIB,DC=ST,MF=LP.
DISPOSE,DN=$SCILIB,DC=ST,MF=LP.
DISPOSE,DN=SKOL,DC=ST,MF=PP.
DISPOSE,DN=CAL,DC=ST,MF=PP.
DISPOSE,DN=LDR,DC=ST,MF=PP.
DISPOSE,DN=UPDATE,DC=ST,MF=PP.
DISPOSE,DN=BUILD,DC=ST,MF=PP.
DISPOSE,DN=CFT,DC=ST,MF=FP.
DISPOSE,DN=UNB,DC=ST,MF=CP.
*.
*.        DISPOSE ALL NEW PROGRAM LIBRARIES
*.        THE PROGRAM LIBRARIES ARE TO BE DISPOSED ONLY AFTER THE REST
*.        OF THE JOB HAS RUN SUCCESSFULLY
*.
DISPOSE,DN=FTLIBNL,SDN=FTLIBPL,DC=ST,MF=LP.
DISPOSE,DN=SCILBNL,SDN=SCILBPL,DC=ST,MF=LP.
DISPOSE,DN=SYSLBNL,SDN=SYSLBPL,DC=ST,MF=LP.
DISPOSE,DN=PRODNL,SDN=PRODPL,DC=ST,MF=PP.
DISPOSE,DN=CFTNL,SDN=CFTPL,DC=ST,MF=FP.
DISPOSE,DN=COSNL,SDN=COSPL,DC=ST,MF=CP.
*.
*.
*.
EXIT.
/EOF
*C ISAMAX,ICAMAX,SASUM,SCASUM,SAXPY,CAXPY,SCOPY,CCOPY,SDOT,CDOT
*C SNRM2,SCNRM2,SROT,SROTG,SROTM,SROTMG,SSCAL,CSSCAL,CSCAL,SSWAP,CSWAP
*C MXMA,MINV,CRFFT2,RCFFT2,PACK,UNPACK,FILTERS,FILTERG,SSUM,CSUM
*C MXM,SCATTER,GATHER,CROT,CROTG,CFFT2,SCERP,ISMAX,ISMIN
/EOF
*C OPFILT
/EOF
```

DGS SOFTWARE GENERATION 4

4.1 PROCEDURE

The following steps provide for generating the Eclipse local station (STAT.<OL,SV> and STATF.<OL,SV>), the concentrator (CON.<OL,SV> and CONF.<OL,SV>), the remote station (REM.<OL,SV> and REMF.<OL,SV>), the CRAY-1 simulator (SIM.<OL,SV> and SIMT.<OL,SV>), and the CRI-supplied Eclipse utilities.

1. Create a directory for station generation:
   (a) CDIR STAT ↵
   (b) DIR STAT ↵

2. Load procedure, command, and source files from the DGS software release tape:
   (a) Mount the release tape on unit 0.
   (b) INIT MTØ ↵
   (c) DPØ:LOAD/V MTØ:(Ø,1,2) ↵
   (d) Release MTØ

3. Generate the station and simulator using the following procedures:
   (a) LOGON ST ↵
   (b) SUBMIT STATJOB ↵
   (c) GENERATE† ↵

   NOTE

   Wait until STATJOB has completed and transferred station files before typing GENERATE.

4. Generate the Eclipse utilities using the procedure file:
   (a) UTILITY† ↵

5. If desired, save the files on tape using the procedure file:
   (a) LOGON ST ↵
   (b) SUBMIT GETPL ↵
   (c) TAPE† ↵
   (d) LISTING† ↵

† Responds appropriately to any prompts output by the procedure files

4.2 LISTINGS

Listings of the procedure files used in the station generation follow:

4.2.1 GENERATE.MC

```
MESSAGE
MESSAGE " THIS PROCEDURE GENERATES ALL VERSIONS OF THE ECLIPSE"
MESSAGE " STATION AND THE SIMULATOR STATION: "
MESSAGE
MESSAGE "       CON.<OL SV>      BACKGROUND CONCENTRATOR. "
MESSAGE "       CONF.<OL SV>     FOREGROUND CONCENTRATOR. "
MESSAGE "       REM.<OL SV>      BACKGROUND REMOTE STATION. "
MESSAGE "       REMF.<OL SV>     FOREGROUND REMOTE STATION. "
MESSAGE "       STAT.<OL SV>     BACKGROUND LOCAL STATION. "
MESSAGE "       STATF.<OL SV>    FOREGROUND LOCAL STATION. "
MESSAGE "       SIM.<OL SV>      CRAY-1 JOB SIMULATOR. "
MESSAGE "       SIMT.<OL SV>     CRAY-1 SIMULATOR. "
MESSAGE
MESSAGE " THE GENERATION PROCEDURE TAKES 4 HOURS. "
MESSAGE
MESSAGE " JOB STATJOB MUST BE RUN AND COMPLETED BEFORE CONTINUTING"
MESSAGE
MESSAGE/P " IF YOU WISH TO STOP NOW, PRESS CTRL-A.  OTHERWISE"
UNLINK - -
%MDIR%:LINKS
UNLINK MAC.PS
GTOD
DELETE L$-.LS
OUT/S (@L$SOURCE.FL@).<BL,SR>
OUT/S L$GLOBA.BL L$GLOBA.SR
DELETE L$-.BL
MAC/F/L/N L$GLOBA MACE.PS/T
MAC/S     L$GLOBA MACE.PS/T
DELETE L$MAC.PS
RENAME MAC.PS L$MAC.PS
GTOD
MESSAGE
MESSAGE " GENERATING SLA LIBRARY. "
MESSAGE
@SLA.CM@
GTOD
MESSAGE
MESSAGE " ASSEMBLING CONCENTRATOR ROUTINES. "
MESSAGE
DELETE C$-.LS
OUT/S (@C$OUT.FL@).<BL,SR>
OUT/S C$GLOBA.BL C$GLOBA.SR
DELETE C$-.BL
MAC/F/L/N C$GLOBA L$MAC.PS/T
MAC/S     C$GLOBA L$MAC.PS/T
DELETE C$MAC.PS
RENAME MAC.PS C$MAC.PS
@C$LINKS.CM@
```

```
MAC/F/L (@C$SOURCE.FL@) C$MAC PS/T
MAC/F/L SLADF/S (C$CONFI,C$SLA) C$MAC.PS/T
UNLINK C$-.SR
DELETE C$-.SR
GTOD
MESSAGE
MESSAGE " ASSEMBLING REMOTE STATION ROUTINES."
MESSAGE
DELETE R$-.LS
OUT/S (@R$OUT.FL@).<BL,SR>
OUT/S R$GLOBA.BL R$GLOBA.SR
DELETE R$-.BL
MAC/F/L/N R$GLOBA C$MAC.PS/T
MAC/S     R$GLOBA C$MAC.PS/T
DELETE R$MAC.PS
RENAME MAC.PS R$MAC.PS
@R$LINKS.CM@
MAC/F/L (@R$SOURCE.FL@) R$MAC.PS/T
UNLINK R$-.SR
DELETE R$-.SR
GTOD
MESSAGE
MESSAGE " ASSEMBLING CRAY-1 SIMULATOR ROUTINES."
MESSAGE
DELETE T$-.LS
OUT/S (@T$OUT.FL@).<BL,SR>
OUT/S T$GLOBA.BL T$GLOBA.SR
DELETE T$-.BL
MAC/F/L/N T$GLOBA L$MAC.PS/T
MAC/S     T$GLOBA L$MAC.PS/T
DELETE T$MAC.PS
RENAME MAC.PS T$MAC.PS
@T$LINKS.CM@
MAC/F/L (@T$SOURCE.FL@) T$MAC.PS/T
UNLINK T$-.SR
DELETE T$-.SR
GTOD
MESSAGE
MESSAGE " ASSEMBLING JOB SIMULATOR ROUTINES."
MESSAGE
DELETE S$-.LS
OUT/S (@S$OUT.FL@).<BL,SR>
OUT/S S$GLOBA.BL S$GLOBA.SR
DELETE S$-.BL
MAC/F/L/N S$GLOBA T$MAC.PS/T
MAC/S     S$GLOBA T$MAC.PS/T
DELETE S$MAC.PS
RENAME MAC.PS S$MAC.PS
@S$LINKS.CM@
MAC/F/L (@S$SOURCE.FL@) S$MAC.PS/T
UNLINK S$-.SR
DELETE S$-.SR
GTOD
MESSAGE
MESSAGE " ASSEMBLING STATION ROUTINES."
```

```
MESSAGE
MAC/F/L (@L$SOURCE.FL@) L$MAC.PS/T
GTOD
MESSAGE
MESSAGE " LOADING LOCAL STATION."
MESSAGE
@L$STAT.CM@
@L$STATF.CM@
DELETE L$-.SR
GTOD
MESSAGE
MESSAGE " LOADING REMOTE STATION"
MESSAGE
@R$REM.CM@
@R$REMF.CM@
GTOD
MESSAGE
MESSAGE " LOADING CONCENTRATOR "
MESSAGE
@C$CON.CM@
@C$CONF.CM@
GTOD
MESSAGE
MESSAGE " LOADING CRAY-1 SIMULATOR"
MESSAGE
@T$SIM.CM@
GTOD
MESSAGE
MESSAGE " LOADING JOB SIMULATOR"
MESSAGE
@S$SIM.CM@
MESSAGE
MESSAGE " SIMULATOR  AND STATION GENERATION IS COMPLETE."
MESSAGE
```

4.2.2 UTILITY.MC

```
DELETE UTILITY.SC
MESSAGE
MESSAGE " THIS PROCEDURE GENERATES ALL OF THE ECLIPSE UTILITIES:"
MESSAGE
MESSAGE "       BLOCK     CAL       DMP       ED        OUT"
MESSAGE "       RDF       READ      RESET     UPDATE    XFERC"
MESSAGE "       AOSLOD    CVD"
MESSAGE
MESSAGE " JOB STATJOB MUST BE RUN BEFORE CONTINUTING"
MESSAGE
MESSAGE/P " IF YOU WISH TO STOP NOW, PRESS CTRL-A.  OTHERWISE "
MESSAGE
UNLINK - -
%MDIR%:LINKS
OUT/S (@UTILOUT.FL@).<BL,SR>
DELETE (@UTILOUT.FL@).LS
MESSAGE
MESSAGE " ASSEMBLING ED"
MESSAGE
MAC/F/L ED MACE.PS/T
UNLINK EDF.SV
DELETE EDF.MP
RLDR/A/E/P 254/Z 17000/F ED EDF.SV/S ED.MP/L
MESSAGE
MESSAGE " ASSEMBLING XFERC"
MESSAGE
MAC/F/L XFERC MACE.PS/T
UNLINK XFERC.SV
DELETE XFERC.MP
RLDR/A/E/P XFERC XFERC.MP/L
MESSAGE
MESSAGE "ASSEMBLING OUT"
MESSAGE
MAC/F/L OUT MACE.PS/T
MAC/F/L CVD MACE.PS/T
UNLINK OUT.SV
DELETE OUT.MP
RLDR/A/E/P OUT CVD OUT.MP/L
MESSAGE
MESSAGE " ASSEMBLING READ"
MESSAGE
MAC/F/L READ MACE.PS/T
UNLINK READ.SV
DELETE READ.MP
RLDR/A/E/P READ READ MP/L
MESSAGE
MESSAGE " ASSEMBLING CAL "
MESSAGE
```

```
MAC/F/L CAL(,0,1,2,X) MACE.PS/T
UNLINK CAL OL CAL.SV
DELETE CAL.MP
RLDR/A/E/P CAL CALX [CAL0,CAL1,CAL2] CAL.MP/L
MAC/F/L CALCREF MACE.PS/T
UNLINK CREF.SV
DELETE CALCREF.MP
RLDR/A/E/P CALCREF CREF.SV/S CALCREF.MP/L
MESSAGE
MESSAGE " ASSEMBLING BLOCK "
MESSAGE
MAC/F/L BLOCK MACE.PS/T
UNLINK BLOCK.SV
DELETE BLOCK.MP
RLDR/A/E/P BLOCK BLOCK.MP/L
MESSAGE
MESSAGE " ASSEMBLING UPDATE "
MESSAGE
MAC/F/L UPDATE MACE.PS/T
UNLINK UPDATE.SV
DELETE UPDATE.MP
RLDR/A/E/P UPDATE UPDATE.MP/L
MESSAGE
MESSAGE " ASSEMBLING DMP"
MESSAGE
MAC/F/L DMP MACE.PS/T
UNLINK DMP.SV
DELETE DMP.MP
RLDR/A/E/P DMP DMP.MP/L
MESSAGE
MESSAGE " ASSEMBLING AOSLOD"
MESSAGE
MAC/F/L AOSLOD MACE.PS/T
UNLINK AOSLOD.SV
DELETE AOSLOD.MP
RLDR/A/E/P AOSLOD 10/K AOSLOD.MP/L
MESSAGE
MESSAGE " ASSEMBLING RDF"
MESSAGE
MAC/F/L RDF MACE.PS/T
UNLINK RDF.SV
DELETE RDF.MP
RLDR/A/E/P RDF RDF.MP/L
MESSAGE
MESSAGE " ASSEMBLING RESET"
MESSAGE
MAC/F/L RESET MACE.PS/T
UNLINK RESET.SV
DELETE RESET.MP
RLDR/A/E/P RESET RESET.MP/L
DELETE (@UTILOUT.FL@).BL
DELETE (@UTILOUT.FL@).RB
DELETE (@UTILOUT.FL@).SR
MESSAGE
MESSAGE " UTILITY GENERATION IS COMPLETE. "
MESSAGE
```

4.2.3 TAPE.MC

```
MESSAGE
MESSAGE " THIS PROCEDURE GENERATES A TAPE CONTAINING ECLIPSE UTILITY"
MESSAGE " GENERATION FILES."
MESSAGE
MESSAGE "      FILE 0 - PROCEDURE FILES WHICH GENERATE THE ECLIPSE UTIL-"
MESSAGE "               ITIES."
MESSAGE "      FILE 1 - ECLIPSE UTILITIES AND  COMMAND FILES.  THIS FILE"
MESSAGE "               IS A COPY OF FILE 9 ON THE DEADSTART TAPE."
MESSAGE "      FILE 2 - STATION PROGRAM AND UTILITY PROGRAM PL'S."
MESSAGE "      FILE 3 - LOAD MAPS."
MESSAGE "      FILE 4 - LIBRARY FILES, BINARY FILES, AND SYMBOL TABLES,"
MESSAGE "               AND SLA SOURCE FILES."
MESSAGE "      FILE 5 - A COPY OF FILE 0."
MESSAGE "      FILE 6 - A COPY OF FILE 1."
MESSAGE "      FILE 7 - A COPY OF FILE 3."
MESSAGE "      FILE 8 - A COPY OF FILE 4."
MESSAGE
MESSAGE "      ASSEMBLY LISTINGS ARE ON A SEPERATE TAPE."
MESSAGE
MESSAGE " TAPE GENERATION TAKES 20 MINUTES."
MESSAGE
MESSAGE " IF YOU WISH TO STOP NOW, PRESS CTRL-A.  OTHERWISE"
EQUIV/P TAPE MT0
INIT TAPE
GTOD
DUMP/A/K/V TAPE:0 C$CON.CM C$CONF.CM C$LINKS.CM C$OUT.FL R$OUT.FL ^
                  C$SOURCE.FL GENERATE.MC UTILOUT.FL STATJOB ^
                  L$SOURCE.FL L$STAT.CM L$STATF.CM LISTING.MC ^
                  R$LINKS.CM R$REM.CM R$REMF.CM R$SOURCE.FL ^
                  S$LINKS.CM S$SIM.CM S$SOURCE.FL T$OUT.FL S$OUT.FL ^
                  SLA.CM T$LINKS.CM T$SIM.CM T$SOURCE.FL ^
                  TAPE.MC UTILITY.MC GETPL
DUMP/A/K/V TAPE:1 @CRI.FL@
DUMP/A/K/V TAPE:2 STATPL
DUMP/A/K/V TAPE:3 -.MP
DUMP/A/K/V TAPE:4 -.RB -.LB -.PS SLA-.SR
DUMP/A/K/V TAPE:5 C$CON.CM C$CONF.CM C$LINKS.CM C$OUT.FL R$OUT.FL ^
                  C$SOURCE.FL GENERATE.MC UTILOUT.FL STATJOB ^
                  L$SOURCE.FL L$STAT.CM L$STATF.CM LISTING.MC ^
                  R$LINKS.CM R$REM.CM R$REMF.CM R$SOURCE.FL ^
                  S$LINKS.CM S$SIM.CM S$SOURCE.FL T$OUT.FL S$OUT.FL ^
                  SLA.CM T$LINKS.CM T$SIM.CM T$SOURCE.FL ^
                  TAPE.MC UTILITY.MC GETPL
DUMP/A/K/V TAPE:6 @CRI.FL@
DUMP/A/K/V TAPE:7 -.MP
DUMP/A/K/V TAPE:8 -.RB -.LB -.PS SLA-.SR
RELEASE TAPE
GTOD
MESSAGE
MESSAGE " TAPE GENERATION IS COMPLETE."
MESSAGE
```

4.2.4 LISTING.MC

```
MESSAGE
MESSAGE " THIS PROCEDURE GENERATES A TAPE CONTAINING UTILITY AND"
MESSAGE " STATION LISTINGS: "
MESSAGE
MESSAGE "       FILE 0 - UTILITY ROUTINES. "
MESSAGE "       FILE 1 - LOCAL STATION. "
MESSAGE "       FILE 2 - SLA ROUTINES. "
MESSAGE "       FILE 3 - CONCENTRATOR. "
MESSAGE "       FILE 4 - REMOTE STATION. "
MESSAGE "       FILE 5 - SIMULATOR. "
MESSAGE
MESSAGE " TAPE GENERATION TAKES 20 MINUTES. "
MESSAGE
MESSAGE " IF YOU WISH TO STOP NOW, PRESS CTRL-A.  OTHERWISE"
EQUIV/P TAPE MT0
INIT TAPE
GTOD
DUMP/A/K/V TAPE: 0 BLOCK.LS CAL<,0,1,2,X,CREF>.LS DMP.LS ED.LS OUT.LS ^
                   RDF.LS READ.LS RESET.LS UPDATE.LS XFERC LS AOSLOD.LS
DUMP/A/K/V TAPE: 1 L$-.LS
DUMP/A/K/V TAPE: 2 SLA-.LS
DUMP/A/K/V TAPE: 3 C$-.LS
DUMP/A/K/V TAPE: 4 R$-.LS
DUMP/A/K/V TAPE: 5 S$-.LS T$-.LS
RELEASE TAPE
GTOD
MESSAGE
MESSAGE " TAPE GENERATION IS COMPLETE. "
MESSAGE
```

Part 3

SYSTEM STARTUP AND RECOVERY

CONTENTS
PART 3 SYSTEM STARTUP AND RECOVERY

1. COS SYSTEM STARTUP 1-1
   1.1 PARAMETER FILE 1-1
      1.1.1 Startup mode parameters 1-1
      1.1.2 Enter memory parameters 1-2
      1.1.3 Breakpoint selection parameters 1-4
      1.1.4 END parameter 1-5
      1.1.5 CRAY-1 memory size parameter. 1-5
      1.1.6 Device parameters 1-6
      1.1.7 Disk flaw parameters 1-10
      1.1.8 Dump control parameters 1-11
      1.1.9 Rolled job recovery parameters. 1-12.1
      1.1.10 Permanent dataset recovery parameters 1-12.1
      1.1.11 System Directory recovery parameter 1-12.2
      1.1.12 Job class structure parameters. 1-12.2
      1.1.13 Boot control parameters 1-12.3
   1.2 STARTUP PROCEDURE 1-13
   1.3 ZLOG 1-14
   1.4 SYSTEM LOG MESSAGES 1-16
2. SDR INITIALIZATION 2-1
3. SYSTEM DUMP PROCESSING 3-1
   3.1 SYSTEM DUMPING. 3-1
      3.1.1 DUMP - Dump through Data General Station. 3-1
      3.1.2 DDC - Dump to CRAY-1 disk 3-2
   3.2 SYSTEM DUMP FORMAT 3-3
      3.2.1 Dump format from Data General dump. 3-3
      3.2.2 Dump format from dump to disk 3-4

4. RECOVERY OF ROLLED JOBS 4-1
4.1 INTRODUCTION. 4-1
4.2 ROLLED JOB INDEX DATASET. 4-1
4.3 RRJ SUBROUTINE 4-4
4.3.1 RRJ execution during Install. 4-4
4.3.2 RRJ execution during Deadstart. 4-5
4.3.3 RRJ execution during Restart. 4-5
4.4 JOB RECOVERY. 4-6
4.4.1 Index entry validation. 4-6
4.4.2 Roll dataset validation 4-6
4.4.3 DAT validation. 4-7
4.4.4 Dataset reservation 4-8
4.4.5 Pseudo-access of permanent datasets 4-9
4.4.6 Resource deallocation 4-9
4.4.7 Job recovery completion 4-9
4.5 TERMINATION OF RRJ. 4-10
4.6 MESSAGES FROM RRJ 4-10
4.6.1 Messages during $ROLL processing. 4-10
4.6.2 Job-related messages. 4-11
5. RECOVERY OF THE SYSTEM DIRECTORY 5-1
5.1 INTRODUCTION. 5-1
5.2 SYSTEM DIRECTORY DATASET. 5-1
5.3 SDRREC SUBROUTINE 5-2
5.3.1 File allocation 5-2
5.3.2 SDR recovery. 5-2
5.3.3 No recovery specified 5-3
5.3.4 Changes in the number of SDR entries. 5-3
5.4 MESSAGES FROM SDR RECOVERY. 5-3
5.4.1 Messages issued by SDR recovery 5-3

COS SYSTEM STARTUP 1

1.1 PARAMETER FILE

A parameter file must reside on the Eclipse disk in order for the COS system to be started via the STARTUP command issued at the Eclipse station.

The parameter file is generated under RDOS by the operator or an analyst and contains one record per parameter. One method of generation could be to punch the parameters onto cards and then transfer the cards to an Eclipse file. The STARTUP command assumes that the name of this file will be COSPAR, by default.

A parameter record consists of an asterisk, a parameter verb (except for memory entry parameters), and arguments as required by the verb. Arguments are separated from each other and from the verb by commas. Except for *asciidata* (see section 1.1.2), the first space encountered is considered the start of a comment and ends the effective portion of the parameter record. Each parameter record, including any comment portion, is terminated by an ASCII carriage return code, 015_8.

1.1.1 STARTUP MODE PARAMETERS

Formats:

*INSTALL

*DEADSTART

*RESTART

These parameters are mutually exclusive. Each selects one of the startup modes. The last parameter encountered is used. None of these parameters has any arguments. The default startup mode if no parameter file is present is determined by an installation parameter. If there is a parameter file, however, one of these parameters must be specified.

For *INSTALL, COS is started as if for the very first time. All CRAY-1 mass storage is assumed to be vacant and the DSC and device labels are initialized.

For *DEADSTART, COS is started as if after a normal system shutdown.

For *RESTART, COS is started as if after a system interruption.

Jobs that were in execution and the job class structure that was in effect at the time of the interruption may be recovered in progress at operator option.

1.1.2 ENTER MEMORY PARAMETERS

The parameter file can contain any number of the following parameters presented in the forms noted. If two or more parameters change the same parcel, the last one encountered is valid.

**addr*	Stores 64-bits of zero at address addr.
**addr,data*	Stores octal value right adjusted with zero fill at word address addr.
**addr,L data*	Stores octal value left adjusted with blank fill at word address addr.
**Aaddr,asciidata*	Stores ASCII data right adjusted with zero fill at word address addr.
**addr,p,data*	Stores octal value right adjusted with zero fill at parcel p of word addr.

addr Word address in octal. However, if the characters A, B, C, or D appear in the address, they are each interpreted as a 2-bit binary value of 00,01,10, and 11, respectively. The occurence of A, B, C, or D in the address does not designate a parcel address. A command consisting of only the address causes zeros to be stored in the word.

data	Octal data. For convenience in entering parcel addresses, the characters A, B, C, and D can appear in the data and are interpreted as binary 00, 01, 10, and 11, respectively. For word entries, bits beyond the 64th are truncated. For parcel entries, bits beyond the 16th are truncated. Truncation is from left to right. Fewer than 64 bits (or 16 bits for a parcel) are stored right adjusted with zero fill.
L*data*	Resembles data; however, the presence of L before the octal data causes data to be stored left adjusted with zero fill. The leftmost digit must be 0 or 1.
asciidata	A string of not more than 8 ASCII characters. ASCII data is stored right adjusted with zero fill. Data is interpreted as ASCII characters when an A precedes the word address.
p	Parcel designator (A, B, C, or D). Only 16 bits of octal data will be stored. The comma separating the designator from the address is required. Left adjust and ASCII data features are not supported.

1.1.3 BREAKPOINT SELECTION PARAMETERS

There exist two methods of setting breakpoints during system Startup: *EBP and *DEBUG. The *DEBUG command is specifically intended for use in debugging Startup; *EBP may be used to debug any portion of STP, including Startup.

*EBP sets a breakpoint in any STP-relative parcel address. This may include any address within Startup.

Format:

*EBP,*n*,*paddr*

Parameters:

n	Number of breakpoint (0-7)
paddr	Parcel address relative to STP; must be terminated by A, B, C, or D

*DEBUG allows Startup to calculate and set a breakpoint automatically. This breakpoint is set at the entry point to the specified Startup option (Install, Deadstart, or Restart) and may be used to halt Startup without requiring changes to the parameter file each time the system is reassembled. The S registers contain an ASCII message informing the operator that the breakpoint was set due to a *DEBUG card rather than due to parameter file errors.

Format:

*DEBUG

The *DEBUG command performs the same function as the following command, where nnnnp equals the address of ZINSTALL, ZDEAD, or ZRESTART:

*EBP,1,nnnnnp

*EBP and *DEBUG may be used in the same parameter file, but if *DEBUG is present, breakpoint number 1 may not be used on a *EBP parameter within the same parameter file.

Note that when breakpoints are placed in Startup, some special problems arise. Since breakpointed code must be restarted by commands from the operator station, the breakpoint must occur after SCP has been initialized. It is then possible to log on while Startup is waiting at a breakpoint. Because SCP allows output datasets to be transferred to the station even though Startup is not complete, STAGE OFF must be specified at LOGON while Startup is breakpointed. When the transferred dataset is deleted, the Permanent Dataset Manager attempts to place a message in the system log; however, the log task has not been created, since Startup has not completed. As a result, the system crashes calling an unknown task. This same result can occur through operator command (i.e., LIMIT or DATASET). When Startup is at a breakpoint, only the debug functions should be used.

1.1.4 END PARAMETER

The *END parameter signals the end of the parameter file. It has no arguments. *END is a required parameter.

Format:

```
*END
```

1.1.5 CRAY-1 MEMORY SIZE PARAMETER

The *MEMSIZ parameter overrides the I@MEM installation parameter but must not exceed the actual memory size or I@MEM.

Format:

*MEMSIZ,*memory size in words*

1.1.6 DEVICE PARAMETERS

Device parameters are used to define attributes of peripheral devices and to specify what action Startup will take when permanent datasets or I/O queue datasets are found residing on specific devices. Device parameters allow a device to be logically deleted from the system, logically added to the system, and designated as read-only. They may also be used to delete permanent datasets residing on a device, or to retain a dataset but flag it as inaccessible.

Device allocation control

Two parameters control subsequent allocation of datasets to a device: *ON and *OFF.

The *ON parameter declares a device allocatable. This means that during operation of the system, the Disk Queue Manager allocates new datasets onto the device in accordance with the queue manager allocation algorithm.

Format:

*ON,ldv_1,ldv_2,...,ldv_n

The *OFF parameter is used to declare a device non-allocatable. This means that during operation of the system, the Disk Queue Manager will not allocate any new datasets onto the device.

Format:

*OFF,ldv_1,ldv_2,...,ldv_n

If an *ON and an *OFF parameter reference the same device name, the last parameter encountered is used. The master device may be referenced by *ON but not by *OFF.

Device availability control

The *DOWN, *RELEASE, and *UP parameters control the availability of devices. A device may be configured (meaning the device has an entry in the Equipment Table and a Disk Reservation Table) when it is not physically present. Such a device must be declared unavailable. (Similarly, a device that was previously not available may become available.) A device that was once available and has become unavailable may contain permanent datasets or I/O queue datasets. These datasets may be retained even though the device is not available, or they may be discarded.

The *DOWN parameter declares a device unavailable. The device remains in the EQT and has a DRT, but no datasets will be allocated onto the device and any datasets already on the device will be flagged in the DSC entry as inaccessible. The Permanent Dataset Manager will not permit users to use the ACCESS control statement to access these datasets. The master device may not be referenced in a *DOWN command.

Format:

*DOWN,ldv_1,ldv_2,...,ldv_n

The *RELEASE parameter causes the Dataset Catalog entries for a dataset that resides on a named device (whether wholly or in part) to be deleted by Startup. The device may or may not also be referenced by a *DOWN parameter. The master device may not be referenced by a *RELEASE parameter.

Format:

*RELEASE,ldv_1,ldv_2,...,ldv_n

The *UP parameter declares a device available during a Deadstart or Restart when the device was previously not available. It may be used to add a device not previously present or to make a device available after hardware problems (e.g., bad cables) that previously prevented access to the device have been remedied.

Format:

*UP,ldv_1,ldv_2,...,ldv_n

The *UP parameter for the device takes precedence over a previous *DOWN in the same parameter file but a subsequent *DOWN for the device is illegal. *UP cannot be the default and cannot be specified for the device identified as the master device in the EQT.

If a device becomes unavailable because of problems unrelated to data, such as bad cables, power supply, etc., it may be downed by a *DOWN parameter on the next Restart. When the problem has been corrected, removing the *DOWN parameter from the parameter file makes the device available again. If the device label is no longer present, the *UP command is required to make a device available again.

Examples:

Assume that device DD-19-30 is configured, and has been available. Due to hardware problems, DD-19-30 must be removed from the system. Later, the hardware problems are corrected, and the DD-19-30 is put back into use.

1) Startup parameter file before hardware problem

```
*RESTART
*FLAW,DD-19-30
C5,T4
*ENDFLW
*FLAW,DD-19-20
C241,T7-10
*ENDFLW
*END
```

2a) Startup parameter file to make device unavailable.

```
*RESTART
*DOWN,DD-19-30
   :
   :
*END
```

Note that datasets allocated to the device remain in the DSC, but may not be accessed by users.

2b) Startup parameter file to make device unavailable.

```
*RESTART
*DOWN,DD-19-30
*RELEASE,DD-19-30
   :
   :
*END
```

Note that datasets allocated to the device are deleted from the DSC. This format should be used if the hardware problems are such that data errors are likely (head crash, scratched pack, etc.)

2c) Startup file to leave device available for reading.

```
*RESTART
*OFF,DD-19-30
   :
   :
*END
```

This format allows the operator to attempt to dump datasets that reside on the device. Normally, this would be followed by a Startup using the parameter file in (2b) to delete the datasets after they have been dumped.

3) Startup parameter file to restore DD-19-30.

a) If the data on the pack is still valid, it is possible to restore the device simply by removing the *DOWN parameter file entry. The DSC flags for datasets on the device will be cleared to allow access from user jobs.

b) If the data on the pack is believed to be invalid, or if the device label cannot be found, the pack can be relabeled and restored to the system by the following parameter file:

```
*RESTART
*UP,DD-19-30
*RELEASE,DD-19-30
   :
   :
*END
```

*RELEASE will delete any datasets that are in the DSC and reside on DD-19-30. *UP will allow Startup to re-label the device without doing an Install-type Startup. The *RELEASE should be removed before the next Startup.

1.1.7 DISK FLAW PARAMETERS

Three parameters provide for reserving disk tracks that have flaws or for reserving tracks for engineering diagnostic use. The parameters must appear in sequence and once the sequence begins no other parameter is legal until the sequence ends. An illegal parameter in the sequence will be treated as an unrecognized keyword. The required sequence is:

```
*FLAW,ldv
<flaw cards>
*ENDFLW
```

In the *FLAW parameter, *ldv*, identifies the device to which all subsequent flaw cards apply until a *ENDFLW parameter is encountered.

The *flaw cards* following the *FLAW parameter can be in the following formats:

C*nnn*	Flaw all of cylinder *nnn*.
C*nnn*-*mmm*	Flaw cylinders *nnn* through *mmm*, inclusively.
C*nnn*,T*mm*	Flaw track *mm* of cylinder *nnn*.
C*nnn*,T*mm*-*pp*	Flaw track *mm* through track *pp* of cylinder *nnn*.

Flaw cards are recognized only between *FLAW and *ENDFLW parameters. Flaws for a device can be in any order. All numbers are octal.

Normal processing of parameters resumes following the *ENDFLW parameter.

These parameters provide for adding flaws encountered during system operation without requiring reassembly of the system. Note that currently, the addition of a flaw does not affect a permanent dataset that may occupy a flawed track. Prevention of further read errors on a flawed dataset requires that the dataset be recreated after the flaw is added. No new dataset will be allocated on the flawed track. Note also that setting a flaw onto a track that has a permanent dataset allocated to it can be negated if the permanent dataset is deleted following the Startup.

1.1.8 DUMP CONTROL PARAMETERS

Two parameters control system dump processing during Startup: *DUMP and *NODUMP. *DUMP forces Startup to create a saved dataset containing the image of the preallocated system dump area. *NODUMP inhibits Startup from processing the system dump. Note that *DUMP and *NODUMP may appear in the same parameter file without any error indication, but *NODUMP will take precedence.

Ordinarily, when a system dump is written to the pre-allocated area of disk, a flag is set in the first sector of the dataset, indicating that a new (uncopied) system dump exists. During the next Restart or Dead-start, this flag will be recognized and a copy of the pre-allocated area will be made and saved as a permanent dataset. After successfully

saving the copy of the dump, Startup rewrites the sector to clear the flag so that subsequent Startups will not copy the same dump again. If the user wishes to make another copy of the dump or if the flag does not get set properly during the dump process, *DUMP may be used to cause Startup to unconditionally copy and save the dataset even though the flag is clear in the pre-allocated area.

Format:

```
*DUMP
```

The *NODUMP command inhibits Deadstart and Restart from attempting to recover the pre-allocated area or copy a dump.

When a system including the system dump feature is installed, the pointer in the device label is set and a system dump area is allocated on disk. The first sector of the allocated area contains a list of allocation units set aside for the system dump.

If, following the installation of a system including the system dump feature, a system without the feature is recovered, the disk space for the dump will no longer be reserved in the DRT and the AI list may be overwritten during operation of the new system. Then, Startup of a system including the dump feature would be unable to recover the dump area correctly, even though the device label pointer exists. Under such conditions, Startup may be unable to complete without inhibiting dump recovery; once dump recovery is inhibited, it must always be inhibited until after the next INSTALL.

Format:

```
*NODUMP
```

1.1.9 ROLLED JOB RECOVERY CONTROL PARAMETERS

When a Restart is selected, it is possible for the operator to also select recovery of rolled jobs. For any other option, recovery cannot be selected; if an attempt is made to select recovery for either a Deadstart or Install, the parameter is ignored and a message is sent to the system log indicating that recovery was not possible.

Recovery is selected by the parameter:

*RRJ,*n*

where *n* represents the level of recovery to be performed. Presently, only n=1 or n=0 is supported. If n=1, recovery is attempted If n=0, no recovery is performed.

1.1.10 PERMANENT DATASET RECOVERY PARAMETERS

When either a Restart or a Deadstart is selected, Startup may encounter permanent datasets in the DSC whose entries contain catastrophic errors such as incomplete DSC entries; entries with bad allocations, indexes or device names; or invalid text lengths. Depending on the installation selected option, Startup either deletes any such datasets from the DSC or leaves the datasets in the DSC and flags them to prevent them from being accessed. Any mass-storage space already reserved for the dataset remains allocated until the next Startup. An SDT entry already allocated is deleted.

The operator may override the installation selected option during Startup. Normally, this would be done to clean up the DSC after the cause of the problem has been determined. This is done by the parameter:

*DSCERR, {DELETE / RETAIN}

DELETE — Invalid datasets are to be removed from the DSC;

RETAIN — Invalid datasets are to remain in the DSC with an error flag set in the DSC entry.

1.1.11 SYSTEM DIRECTORY RECOVERY PARAMETER

Whenever a Restart or Deadstart is selected, the operator may request that the System Directory not be recovered by specifying this parameter. If the System Directory Recovery parameter is not specified, the operating system recovers the resident System Directory by processing records from the $SDR file.

A new edition of $SDR is allocated, and any datasets to be added to the System Directory must be re-entered when this parameter is specified. The parameter is:

*SDR

1.1.12 JOB CLASS STRUCTURE PARAMETER

The operator can invoke a new job class structure when either a Deadstart or a Restart without recovery of rolled jobs is selected. This is done by the parameter:

*JCLASS, *filename*

Parameter:

filename The name of the dataset containing the job class structure definition to be invoked. If the dataset does not exist or cannot be read, an appropriate system log message is issued, and the default job class structure goes into effect.

1.1.13 BOOT CONTROL PARAMETERS§

For either a Deadstart or a Restart, Startup may startup a system that resides on CRAY-1 local disk. The system must be a valid user permanent dataset that is accessible to Startup.

*SYSTEM tells Startup on which dataset the system resides.

Format:

*SYSTEM,PDN=*pdn*[,ID=*user-id*][,ED=*edition*][,RD=*read*]

Parameters:

pdn	The permanent dataset name. This is a required parameter.
user-id	The user identifier (optional).
edition	The edition of the dataset. If omitted, the highest edition is used.
read	Read permission control word (optional).

This parameter may be present during single pass startups.

The *BOOT parameter indicates to Startup that the current system should be discarded and replaced by the system defined by the *SYSTEM parameter. If no *SYSTEM parameter is found in the parameter file, Startup halts when it completes processing the parameter file.

Format:

*BOOT

This parameter is changed to *NOOP for pass 2.

---

§ Deferred implementation

1.2 STARTUP PROCEDURE

1. Generate a parameter file using the parameters defined in the preceding sections.

   Example of parameter deck:

```
*DEADSTART
*ON,DD-19-30,DD-19-31
*FLAW,DD-19-30
C27,T4
C37,T4-6
*ENDFLW
*FLAW,DD-19-31
C403,T2
*ENDFLW
*UP,DD-19-32
*FLAW,DD-19-32
C12,T11
*ENDFLW
*DOWN,DD-19-33
*RELEASE,DD-19-33
*END
```

   This deck can be copied to the ECLIPSE disk from the card reader by issuing the following RDOS command:

   XFER/A $CDR *cospar*/R ↵

   where *cospar* is the name of the parameter file. This file must be a direct access file; that is, it must have the D attribute obtained by appending a /R local flag to the file name.

2. Bring up the ECLIPSE station as described in part 3, section 1 or in the DGS Station Operator's Guide, CRI publication 2240006.

3. At the 455 data screen, enter the following command:

STARTUP,*cos*,*cospar*

cos Name of the Eclipse file containing the COS operating system in binary form.

cospar Name of an Eclipse file containing COS parameters.

The default for *cos* is COS; the default for *cospar* is COSPAR.

1.3 ZLOG

The Startup program, Z, maintains a message log for recording errors encountered while checking parameters and errors encountered dynamically during the startup process. If errors are encountered during parameter processing, a special breakpoint is set at the entry to the Z subroutine. When Z is entered, Startup halts at this breakpoint. The Startup exchange package contains an ASCII message in the S registers informing the operator that Startup has halted at a breakpoint due to errors in ZY (the parameter checking routine). The operator may then examine ZLOG to determine the nature of the errors. If the errors are minor, Startup may be instructed to continue by entering the command:

RUN,TASK,Ø

Otherwise, the operator must correct the parameter file and re-initiate the Startup process. To view the Startup exchange package, the operator may enter the following sequence of commands:

```
ASSIGN,TASK,Ø↵   (↵= carriage return)
DIS,Y,X,,T↵      (this is the default)
Y.↵              (note: the period is required)
```

The contents of ZLOG can be dumped or displayed. To determine the location of ZLOG, look in the low-order area of STP for the STPDD pointers. One of these pointers gives the ZLOG location.

Messages in ZLOG are in ASCII. All messages pertaining to parameter checking identify the character location of the last character scanned. The address in a parameter checking message is a character address. That is, the low order digit of the address identifies the character after the last scanned character, where characters in a word are numbered 0 through 7 from left to right. The remaining digits of the address form the word address relative to STP origin.

The following messages may be issued by Z:

ldv DEVICE LABEL NOT FOUND.

Device label ldv not where expected.

BAD ADDRESS.

Address is missing or exceeds available memory. For EBP, the second address is required if its comma is present.

BAD BREAKPOINT NO.

Breakpoint number missing or out of range.

BAD DATA SPECIFICATION

Memory entry command contains bad data to be entered.

BAD END OF FILE.

End of parameter file was encountered before *END parameter.

BAD END OF LINE.

Parameter is not terminated by a carriage return or by a space, optional comments, and carriage return.

BAD FORMAT ON FLAW CARD

Non-octal character specified for cylinder or track or a C or T not present where expected during *FLAW/*ENDFLW sequence.

BAD MEMORY SIZE.

Memory size parameter missing or out of range.

BAD PARAMETER VALUE

An invalid option was specified or a keyword parameter was followed by an out-of-range value.

ERROR COUNT OTHER THAN STARTUP.

Total number of errors other than those encountered during checking errors.

ILLEGAL SINCE MASTER DEVICE.

Parameter not legal for the master device.

MISSING COMMA.

Required parameter is missing.

MISSING STARTUP OPTION.

*INSTALL or *DEADSTART or *RESTART must be present.

ONLY FLAW CARD ALLOWED

First character of card following *FLAW must be C, or *ENDFLW must appear.

PARAMETER ERROR COUNT.

The total count of parameter checking errors encountered in the parameter file.

RLS MUST NOT BE DEFAULT.

Operating system was incorrectly assembled with release flag set. Use *RELEASE parameter to set the flag.

UNKNOWN LOGICAL DEVICE.

Specified logical device name not found in equipment table.

UNKNOWN PARAMETER TYPE.

Keyword unintelligible to system.

1.4 SYSTEM LOG MESSAGES

In addition to the messages placed in ZLOG, Startup also places messages in the system logfile. This is done by stacking messages at the end of the table area used by Z and extending toward the highest available memory address. At the end of Startup, any messages that have been stacked are issued to the system log by calling the log manager task until all stacked messages are exhausted. If too many messages are required and all of the available memory is used for stacking messages, a special message is placed at the end of the buffer (overwriting the last message in the buffer, if necessary); all subsequent attempts to stack a message are ignored and no error indication is given. The system utility EXTRACT may be used to list these messages (refer to part 4, section 7).

Messages that may be issued during Startup are:

pdn-info DATASET RESIDES ON DOWN DEVICE *ldv*

The dataset identified by *pdn-info* contains at least one reference in its DAT to a device that is indicated in the Equipment Table as being down, but the release flag for that device is not set. The dataset is flagged in the DSC as inaccessible, but is not deleted.

The last device encountered in the DAT that has the down flag set is identified by *ldv*.

pdn-info DATASET RESIDES ON MISSING/RELEASED DEVICE *ldv*

The dataset identified by *pdn-info* contains at least one reference in its DAT to a device for which the EQT entry shows the release flag set. The dataset is deleted from the DSC. The last device encountered in the DAT that has the release flag set is identified by *ldv*.

pdn-info DATASET DAT CONTAINS A1 CONFLICT AT TRACK *trnum* ON *ldv*

The DAT for the dataset references at least one track that is also referenced by at least one additional dataset, or that was declared flawed. The first such track and its device name are identified in the message. Additional conflicts may exist; they are recognized but not identified in the log.

pdn-info DSC ENTRY CONTAINS CATASTROPHIC ERROR - ENTRY *option*

The DSC entry for the dataset contains at least one catastrophic error. Depending on the installation-selected option or the presence of a *DSCERR parameter, the dataset DSC entry is either deleted or flagged as in error and the *option* in the message is deleted or retained accordingly.

DATASET CATALOG ALLOCATES RESERVED TRACK *trnum* ON *ldv*

The DSC as defined in the master device label is reserving a DRT bit that is already set. This usually means a flaw has been added since the previous INSTALL. Each such conflict will be identified in the log.

SYSTEM DUMP AREA ALLOCATES RESERVED TRACK *trnum* ON *ldv*

The preallocated system dump area is reserving a DRT bit that is already reserved. This usually means a flaw has been added since the previous INSTALL. Each such conflict will be identified in the log.

SYSTEM DUMP NOT SAVED - PDM RETURNED STATUS *stat*

The system dump could not be saved. The Permanent Dataset Manager returned an error status at *stat*.

SYSTEM DUMP SAVED - STATUS=*stat* PDN=*pdn* UID=*uid* ED=*ed*

The system dump was successfully saved, with the permanent dataset name, user ID, and edition given in the message.

pdn-info MULTI-TYPE DATASET HAS INCONSISTENT ALLOCATION - QDT INDEX = *nnnn*, ENTRY *option*.

The disk reservation for at least one DSC entry differs from the other DSC entries with the same QDT index. As with catastrophic errors, either I DSCERR or *DSCERR determines whether or not the entry remains in the DSC. If the option is RETAIN, the dataset is inaccessible.

pdn-info MULTI-TYPE DATASET INACTIVATED DUE TO QDT INDEX OUT OF RANGE - INDEX *index*, QDT SIZE=*size*.

A QDT index of a DSC entry pointed beyond the allocated area of the QDT. The dataset remains in the DSC but is flagged as inaccessible. It becomes accessible once a system with a large enough QDT is restarted.

keyword UNKNOWN KEYWORD †

A valid parameter card contains an unknown keyword.

keyword KEYWORD MUST BE EQUATED †

The keyword for a valid parameter card must be associated with a value.

keyword BADLY EQUATED KEYWORD †

The keyword for a valid parameter card is illegal, perhaps because the value is too long, non-numeric, or null.

Additional messages may be issued during an attempt to recover rolled jobs. These messages are described in part 3, section 4.

† Deferred implementation

SDR INITIALIZATION 2

The System Directory (SDR) is memory resident and must be initialized after an Install. SDR entries can be initialized as soon as the datasets to be entered into the directory reside on system mass storage

To enter datasets into the directory, submit a job consisting of ACCESS control statements for which the ENTER parameter is specified. This job should have a high priority so that it runs before any other job in the input queue. This parameter directs COS to enter the dataset name in the system directory. After the dataset has been entered in the directory, it is accessible from any job without the need for ACCESS control statements in each job.

The SDR is recovered during a Restart or Deadstart unless no recovery is specified with the *SDR parameter.

Sample job:

```
JOB,JN=SYSDIR,T=2,M=20,P=15.
*.
*.  SDR initialization job
*.
ACCESS,ENTER,DN=AUDIT.
ACCESS,ENTER,DN=BUILD.
ACCESS,ENTER,DN=CAL.
ACCESS,ENTER,DN=CFT
ACCESS,ENTER,DN=COMPARE.
ACCESS,ENTER,DN=COPYD.
ACCESS,ENTER,DN=COPYF.
ACCESS,ENTER,DN=COPYR.
ACCESS,ENTER,DN=DSDUMP.
ACCESS,ENTER,DN=DUMP.
ACCESS,ENTER,DN=EXTRACT.
ACCESS,ENTER,DN=FDUMP.
ACCESS,ENTER,DN=LDR.
ACCESS,ENTER,DN=PDSDUMP.
ACCESS,ENTER,DN=PDSLOAD.
ACCESS,ENTER,DN=SKIPD.
ACCESS,ENTER,DN=SKIPF.
ACCESS,ENTER,DN=SKIPR.
ACCESS,ENTER,DN=UNB.
ACCESS,ENTER,DN=UPDATE.
ACCESS,ENTER,DN=WRITEDS.
(EOF)
```

SYSTEM DUMP PROCESSING 3

3.1 SYSTEM DUMPING

Following a system failure, the operator has the option of dumping CRAY-1 memory. This may be done in either of two ways: through the Data General station or to CRAY-1 DD-19 disk.

3.1.1 DUMP - DUMP THROUGH DATA GENERAL STATION

The station software that runs on the Data General Eclipse connected to the CRAY-1 MCU channel provides the operator with the ability to dump selected portions of CRAY-1 memory to the disk pack associated with that Eclipse. When this dump completes transferring to the Eclipse disk, the operator may issue a START command to reinitialize COS in the CRAY-1 memory. The operator then has the option of immediately printing the dump or staging it into COS, where it can be made a permanent dataset and can be printed by system utilities. The command to dump CRAY-1 memory to the Eclipse disk is:

DUMP,*filename*,*fwa*,*lwa*

Parameters:

filename	Data General filename to associate with the dump
fwa	Beginning CRAY-1 memory address (absolute)
lwa	Ending CRAY-1 memory address (absolute)

To send this dump in to be saved as a CRAY-1 permanent dataset, use:

SAVE,*filename*,*pdn*

Parameters:

filename	Data General filename to associate with the dump
pdn	CRAY-1 permanent dataset name to be created by COS

Once the dump is resident on the CRAY-1 disk, it may be accessed by a user job that may print selected portions of the dump using system utilities.

3.1.2 DDC - DUMP TO CRAY-1 DD-19 DISK

The DDC utility dumps memory directly from CRAY-1 main memory to the DD-19 disk. The device label for the master device must contain a pointer to the first track of a disk area preallocated to hold a system dump, and the first sector of that track must contain the list of all tracks reserved for the dump. A block of words is reserved in EXEC so that this program may be loaded and executed without overwriting EXEC code or tables. The memory dumped includes all memory from 0 to I@MEM, or until the reserved disk space is filled. A flag is set in the first sector (word 511) to indicate that a new dump exists. During the next Startup, this flag is recognized and a copy of the pre-allocated area is made and saved as a permanent dataset. The flag is then cleared. This permanent dataset may then be accessed by user jobs in order to print selected portions.

Normally, the permanent dataset created by Startup is called CRAY1SYSTEMDUMP; this may be changed by the installation. Under certain conditions, Startup may change the name of the dataset and try to save it again, in response to errors returned by the Permanent Dataset Manager. Errors thus handled include "too many editions" and "maintenance permission not granted." The log message issued when the dataset is successfully saved identifies the name by which the dataset is known to the DSC. When it is necessary to change the name, Startup locates the last non-zero character in the PDN and increments it by one. If the resulting character is alphanumeric, the old name is replaced and the SAVE is attempted again. If the resulting character is not alphanumeric, the original character is left alone and Startup moves one character to the left and repeats the process. If no character in the name can be incremented by one without producing a non-alphanumeric character, Startup is unable to save the dataset and a system log message is issued to that effect.

The DDC utility is invoked by using the SYSDUMP command. The COS generation job stages the absolute binary of the DDC program to the MCU in unblocked format.

Format:

```
SYSDUMP
```

3.2 SYSTEM DUMP FORMAT

System dump datasets created by the DUMP and DDC utilities are in unblocked format, and thus cannot be read by FORTRAN library blocked I/O routines. A utility called FDUMP is provided for processing these dumps. (For a complete description of the FDUMP utility, refer to part 4, section 9.)

3.2.1 DUMP FORMAT FROM DATA GENERAL DUMP

When a dump is staged in from the Data General Eclipse using the sequence below, the permanent dataset created is an unblocked binary dataset, where word zero of the dataset represents *fwa* and the last word of the dataset represents *lwa*, as specified on the DUMP command. There are no headers or trailers in the dataset. This dataset can be printed using the FDUMP utility.

DUMP,*filename*,*fwa*,*lwa*
SAVE,*filename*,*pdn*

3.2.2 DUMP FORMAT FROM DUMP TO DD-19 DISK

When the COS utility DDC is used to create the Deadstart dump, the permanent dataset created is an unblocked dataset having the following format:

Word address	Contents
0-511	Dump header
512-end	CRAY-1 memory dump

The dump header is a 512-word block containing information regarding the dump; the format of the header is given in part 4, section 9 of this manual. Beginning at word 512 of the dataset is an image in unblocked format of CRAY-1 memory at the time of the dump.

RECOVERY OF ROLLED JOBS

4.1 INTRODUCTION

Following any system failure, whether due to software, hardware, or environmental problems, the system operator may choose to attempt to recover any job that may have been in the execution queue at the time of the failure. This section describes the process used to attempt to recover these jobs.

Currently, Startup can successfully recover and restart all of those jobs that were rolled out to mass storage at the time of the system failure, or those that had been rolled out, rolled back in, and performed no additional activity that would cause the roll image on mass storage to be unusable. A job can be recovered if and only if it is certain that the roll image is valid and that repeating any of the activities of the job that may have occurred following roll-in will not cause the results of the job to differ from those that would have been obtained had the failure not occurred. However, permanent datasets accessed following roll-in might not be available following a system recovery if one or more mass storage devices become unavailable; in this event the recovered job receives an error status when attempting to re-access the datasets. Any permanent datasets already accessed by a job prior to rollout must be re-accessed successfully during Startup before the job is considered successfully recovered.

4.2 ROLLED JOB INDEX DATASET

For Startup to be able to determine which jobs were in execution prior to a system recovery, the operating system maintains a special permanent dataset. This dataset, referred to below as $ROLL, contains information about each job that has entered execution and has not yet terminated. $ROLL is maintained in the DSC with a PDN of SYSROLLINDEX; read, write, and maintenance passwords are defined for it. $ROLL is

initialized and saved during an Install. During either a Restart or a Deadstart, subroutine RRJ attempts to access $ROLL. If the ACCESS fails, a new edition of $ROLL is created, initialized, and saved, and recovery of rolled jobs is disabled (with a message to the system log if recovery would have been performed otherwise; no message appears if $ROLL cannot be accessed but recovery was not requested).

The information in $ROLL consists of fixed-length entries, one for each defined JXT entry. The entry corresponding to JXT ordinal zero is used for validation of the $ROLL dataset and does not correspond to any job in the system. Information in entry zero consists of:

- The number of JXT entries defined in the previously deadstarted system. Recovery is not possible if the previous system defined more JXT entries than the current system. An error message is issued in this case.
- The memory size of the previously deadstarted system. This is informational only.
- The logical name of the device containing $ROLL. This is compared with the device name from the DAT that is supplied by the Permanent Dataset Manager when $ROLL is accessed. A mismatch causes an error message to be issued, and recovery is disabled.
- The track number allocated to $ROLL. JXT limitations allow the assumption that $ROLL will never exceed one allocation unit. This number is compared with the AI from the DAT for the accessed $ROLL. A mismatch causes an error message to be issued, and recovery is disabled.
- The sizes of key tables contained in the JTA on the roll index, in particular, LE@RJ, LE@DNT, and LE@JXT. These must be the same in the recovered system or RRJ will halt. RRJ halts rather than continuing with recovery disabled so the operator can restart with a correct system file without having the roll index overwritten.

All other entries in $ROLL correspond to one specific JXT entry. These entries contain enough information to identify which job has been assigned to the JXT entry and to locate the roll image if the job has been rolled

out. The index entry also contains a flag that indicates whether a job has performed some function that will invalidate the roll image. (See the description of the RJ table for detailed descriptions of the formats of these entries.)

Information contained in these entries includes:

- The first three words of the first partition from the DAT for the roll image dataset. This includes the two-word partition header and one word containing up to four allocation unit indices. If the job has never been rolled out, these words are zeroes.
- The job name, job sequence number, station, and terminal ID of job origin. These are used to determine which SDT entry in the input queue corresponds to this job.
- A "not recoverable" flag. This is used to indicate that the job cannot be recovered from the roll image. This flag is set whenever the job performs one of the following functions:
  (1) DELETE, ADJUST, or MODIFY on a permanent dataset. Since these functions change the DSC in a manner that could cause the job to fail if they were repeated, the roll image cannot be relied upon.
  (2) Random write to any dataset. The system CIO routines recognize a random write to a dataset and declare the job not recoverable, since the difference in data may change job results if the job is restarted at an earlier point.
  (3) Write following read, rewind, or skip forward on any dataset. Since a program that reads or skips to end-of-data (or end-of-file) may have different results if the terminator is moved or even removed completely by overwriting, the job is considered not recoverable.
  (4) Release of a local dataset. Since disk space returned to the system is available for use by other jobs, release of a local dataset causes the job to be not recoverable. Release of a permanent dataset does not affect disk allocation and therefore does not affect recoverability.

Note that every job that becomes irrecoverable due to any of the above becomes recoverable again as soon as it has subsequently been successfully rolled out.

$ROLL is maintained during system operation jointly by the Exchange Processor (EXP) and the Job Scheduler (JSH). At job initiation JSH sets up the corresponding index entry to reflect job never rolled out and job not recoverable. Subsequently, each time JSH rolls the job it sets up the index to point to the roll dataset and designates the job to be recoverable. The index is written to disk when the Disk Queue Manager informs JSH that the rollout has completed successfully. EXP recognizes the fact that a job is performing one of the functions that causes the job to become irrecoverable and signals the Job Scheduler to set the index entry accordingly and rewrite the index. The rewrite of the index always occurs before EXP completes processing the function.

4.3 RRJ SUBROUTINE

During Startup, the RRJ subroutine executes as a closed subroutine called by Z. RRJ is called before Z executes SDR recovery and copies any existing system dump, since disk space needed to restart a rolled job must be recovered and allocated in the DRT before any new space can be used. RRJ does not return any status used by Z; it does set a status word indicating the type of recovery performed, which is used by JSH to determine how much JXT initialization JSH must perform.

RRJ performs one of several activities depending on the type of Startup being performed.

4.3.1 RRJ EXECUTION DURING INSTALL

During an Install, recovery of rolled jobs cannot be performed, since permanent datasets and input/output queues are not recovered. Therefore, RRJ merely initializes $ROLL and issues a SAVE request to PDM. The initialization of $ROLL consists of setting up entry zero (see RJ table description in part 5) and zeroing all other entries. The buffer used to write $ROLL remains intact in memory throughout normal operation of the system, and $ROLL is never read during normal operation.

4.3.2 RRJ EXECUTION DURING DEADSTART

Since input/output queues are not recovered during a Deadstart, rolled jobs cannot be recovered either. RRJ attempts to access $ROLL, and reads it into memory. The buffer remains intact throughout normal operation, and $ROLL is never read again during normal operation. If RRJ was enabled by operator specification, RRJ detects that it is a Deadstart, issues an error message, and disables recovery. Once $ROLL has been successfully accessed and read in, the entry zero contents are checked. If errors occur on the access or read, or if entry zero does not validate correctly, RRJ issues error messages and re-initializes $ROLL. A new edition of $ROLL is created if the access was unsuccessful or the existing edition received an error while reading it. Otherwise, the new $ROLL is written over the existing one. If no errors are received, the $ROLL buffer is cleared to indicate no executing jobs and the dataset is rewritten.

4.3.3 RRJ EXECUTION DURING RESTART

If Restart was selected, RRJ may attempt to recover jobs. This depends on whether RRJ is able to successfully access and read $ROLL. Error conditions here are handled as for Deadstart. If the access and read are successful but RRJ was not enabled by the operator (no *RRJ card in the parameter file), then RRJ clears $ROLL as for Deadstart. If RRJ is enabled, RRJ begins scanning the index entries following verification of entry zero. If an error occurs during entry zero validation, RRJ disables recovery with a message to the system log and continues as for a Deadstart.

If no errors occur during $ROLL validation, RRJ attempts to recover jobs. Messages are issued to the system log explaining why a job mentioned in the index is not recovered, or indicating a successful recovery. A successful recovery means that the job has been entered into the JXT chain at the appropriate spot and the input SDT has been moved from the input queue to the execute queue. The job status in the JXT is then

set to rolled-out and suspended-by-recovery. The waiting-for-memory and operator-suspended bits are maintained. All other status bits are set to zero, as are any event wait words. It is the responsibility of any caller who requested recall based on an event to determine for itself whether the event is satisfied or whether the recall should be re-issued. For example, it may be necessary to re-issue any outstanding ACQUIRE requests.

4.4 JOB RECOVERY

Recovery of a job from its latest roll image consists of the steps described below. Any error in any validation step causes the job to become irrecoverable, and an appropriate message is sent to the system log.

4.4.1 INDEX ENTRY VALIDATION

The first step is to validate the information in the index entry. The job cannot be recovered if the index states that the job is irrecoverable, or if the roll dataset is non-existent or resides on a non-existent or unavailable device.

4.4.2 ROLL DATASET VALIDATION

The partition header information in the index entry is used to read in as much of the roll dataset as can be located from the one word of allocation indices contained in the index. It must contain enough of the JTA for the job to be able to locate the copy of the full roll dataset DAT, which was copied to the JTA by the Job Scheduler immediately before rollout along with the JXT image. An error on the read makes the job irrecoverable. Once the first read completes, the JTA size taken from the JTA and from the saved copy of the JXT are compared. An error occurs if the two do not match. This size is then used to determine if more JTA exists. If so, the additional information is read in. Normally, the entire JTA will be read in by the first read, but if many large datasets exist, the JTA may be quite large. RRJ must have the whole JTA in memory at once. It is an error if the JTA does not fit into available

memory above the Startup message stack. The image of the roll DAT is moved from the JTA to the STP DAT area. It is an error if enough DAT pages cannot be allocated in STP to hold the DAT. The roll DAT is then validated. If no errors are found in the DAT, any remaining portion of the JTA and the last block of the user field are read in. They must fit, and there must be no error on the reads. The last block of the user field is located using the JXCJS field of the saved JXT copy. The Job Scheduler stores the current value of the real-time clock in the first block of the JTA and in the last block of the user field immediately prior to rollout. If these do not match, the rollout was only partially complete at the time the system failure occurred.

4.4.3 DAT VALIDATION

Each dataset, including the roll image dataset, must have a zero DAT address in the DNT or must point to a valid DAT. The roll image dataset and the $CS and $IN datasets point to DATs in the STP tables; all others point to DATs in the JTA. To be considered a valid DAT, the following points must be satisfied:

- A multipage DAT must be entirely within the STP tables, or entirely within the JTA. It cannot be in both places.
- A DAT in STP must have DAJORD equal to zero; a DAT in the JTA must have DAJORD equal to the JXT ordinal.
- Successive pages must be numbered correctly.
- A DAT in the JTA must be pointed to by a negative offset that must be within the range indicated by the JTA size; the same is true for each successive page.
- For each partition, the named device must exist and must be available (EQNA must equal zero).
- Each allocation unit index for a partition must be within range for the device.
- For a multitype DAT (DNQDT is nonzero), each allocation unit index must have its corresponding DRT bit set; otherwise an inconsistent allocation has occurred.

- For a DAT that is not multitype (DNQDT is zero), each allocation unit index must not have the corresponding DRT set; otherwise an allocation conflict has occurred.
- When the end of the last page or last partition is reached, the remaining AI count and next partition pointers must be zero.
- When the end of a partition is reached, the next partition pointer (DANPA) must point to either the next word in the current page or the first word following the page header in the next page, or it must be zero.

DAT validation occurs in two passes. The first pass serves as an error scan, and does not set the DRT bits. The second pass actually sets the DRT bits and decrements the available space counts for the device. In this way, RRJ can be sure that a dataset is either completely reserved or completely unreserved. This is necessary for successful deallocation of resources if a later dataset is found to have an error.

4.4.4 DATASET RESERVATION

Each dataset named in the DNT chain in the JTA must be processed. Local datasets must have their DATs validated and the DRT bit maps updated. Permanent datasets must be validated against the Dataset Catalog. Startup will already have updated the DRT bit maps for permanent datasets. PDS entries must also be reconstructed for permanent datasets.

The DNT chain is scanned from the beginning to the end. The memory pool control word preceding each DNT is checked to be sure that the pool entry is in use, and the DNT is checked to ensure that there is a name. If there is no DAT, RRJ goes on to the next DNT. If there is a DAT, and it is in STP (DNDAT is greater than zero) the DNT must be for either $CS or $IN. The SDT entries are searched for an SDT with the correct sequence number, and the DAT address field of the DNT is corrected. RRJ then goes to the next DNT. If the DAT is in the JTA (DNDAT less than zero), the DNT is checked to see if the dataset is permanent. If not, the DAT is validated. If so, a pseudo-access is performed. If no errors are found, RRJ goes to the next DNT. When the end of the DNT scan is reached, the job is considered successfully recovered.

4.4.5 PSEUDO-ACCESS OF PERMANENT DATASETS

When a permanent dataset is encountered in the DNT scan, RRJ requests the Permanent Dataset Manager perform a pseudo-access on the dataset. This process causes the Permanent Dataset Manager to locate the DSC entry for the dataset from the DADSC field of the DAT, and to compare the DAT in the JTA with the DAT in the DSC.

If the DAT appears valid, the Permanent Dataset Manager attempts to construct or update a PDS entry. The DNT permission flags are used to set the PDS permission flags. If the PDS entry already exists, the DNT must indicate read-only permission.

4.4.6 RESOURCE DE-ALLOCATION

If an error occurs at any point in the recovery of a job, any system resources assigned to that job by RRJ must be released. In particular, any disk space reserved for local datasets prior to finding an error on a later dataset, or any PDS entries corresponding to datasets that have already been pseudo-accessed must be de-allocated. For this purpose, the DNT chain is searched once more, until the DNT with the error is reached again. For releasing local datasets, the Disk Queue Manager de-allocate request is used. For releasing PDS entries, the PDM request PMFCRL is used. The disk space for datasets such as $CS or $IN, where the DAT is in STP, is not released. The roll image dataset is released and the STP DAT pages are returned to the system.

4.4.7 JOB RECOVERY COMPLETION

When the end of the DNT chain is reached without error, the job is successfully recovered. The copy of the JXT from the JTA is placed in the JXT area, and the JXT entries are re-linked by priority. The roll image DNT within the JXT is updated to point correctly to the DAT; the SDT entry is moved to the execute queue and the JXT ordinal is placed in the SDT. All wait words are cleared. The JXT status bits are set to R, N, and B (rolled out, not in memory, suspended by recovery) and all other bits except O, A and M (operator suspended, abort pending, and waiting for

memory) are zeroed. The SDT address in the JTA is corrected and the JTA is rewritten to the roll image dataset. A message is sent to the system log and RRJ advances to the next index entry.

4.5 TERMINATION OF RRJ

When the end of the roll index is reached, all entries corresponding to jobs that were not recovered have been cleared. The input queue is scanned, and all jobs that were previously initiated are flagged with a status in the SDT so that CSP will issue log messages when they are re-initiated. Such jobs may be ineligible for rerun, in which case the status passed to CSP reflects that condition; CSP thus terminates the job immediately after issuing the log file messages. The status word RRJSTAT is set to indicate to the Job Scheduler that the JXT entries are already initialized and linked, and the JSH flag SWAPFLAG is set if any jobs were recovered. RRJ then returns to Z.

4.6 MESSAGES FROM RRJ

RRJ issues log messages that:

- Identify the recovered jobs
- Give reasons why unrecovered jobs were not recovered
- Detail any abnormal conditions encountered while processing $ROLL.

These messages are stacked at the end of Startup (as are messages issued while recovering permanent datasets) and are issued to the system log at the end of Startup. Message texts and meanings are as described below.

4.6.1 MESSAGES DURING $ROLL PROCESSING

The following messages are issued during $ROLL processing.

DEADSTART SELECTED - RECOVERY OF ROLLED JOBS DISABLED

Recovery is possible only on a Restart-type Startup.

INDEX AI *num1* MISMATCH ENTRY ZERO AI *num2*

$ROLL DAT shows $ROLL occupies track *num1*, entry zero shows track *num2*.

INDEX DEVICE *ldv* MISMATCH ENTRY ZERO DEVICE *ldv1*

$ROLL DAT shows $ROLL resides on *ldv*, entry zero shows device name *ldv1*.

I/O ERROR ON $ROLL - DQM REPLY WAS *stat*

Disk Queue Manager returned error status *stat* on the read of $ROLL.

OLD SYSTEM JXT COUNT *ct1* GREATER THAN CURRENT SYSTEM COUNT *ct2*

Recovery is not allowed when the previous system defined more JXT entries than the current system.

RECOVERY OF ROLLED JOBS ABORTED

Recovery terminated due to an error. Usually this message is preceded by an explanatory message identifying the error.

UNEXPECTED STATUS ON ACCESS OF $ROLL WAS *stat* - NEW EDITION CREATED - RRJ NOT POSSIBLE

The Permanent Dataset Manager returned error status *stat* on the attempt to access $ROLL. Recovery is not possible. Status 111 (Dataset not found) is never unexpected.

4.6.2 JOB-RELATED MESSAGES

The following are job-related messages.

JOB *jname* - DATASET *dname* DAT VALIDATION ERROR

An error was found in the DAT for a local dataset.

JOB *jname* - DATASET *dname* PDM STATUS *stat* ON PSEUDO-ACCESS

PDM returned error status *stat* when RRJ attempted to access a dataset for the job.

JOB *jname* - DATASET *dname* POINTS TO STP TABLES

Only $CS or $IN may have a DAT in STP.

JOB *jname* - DAT SPACE FULL

Insufficient STP DAT space exists for the roll image DAT.

JOB *jname* - DNT CHAIN BAD

A DNT in the chain has no name, or the memory pool entry is not in use, or the link points outside JTL.

JOB *jname* - INDEX DECLARES NOT RECOVERABLE

The index entry for job jname says the job is not recoverable (RJNRCV equals 1).

JOB *jname* - I/O ERROR *stat* ON ROLL DATASET

DQM returned error status *stat* when RRJ attempted to read the roll image.

JOB *jname* - JTA IMAGE ROLL DAT BAD ORDINAL

The DAJORD field must be zero in all pages of the roll DAT.

JOB *jname* - JTA IMAGE ROLL DAT NEXT PAGE POINTER BAD

The roll image DAT in the JTA must always point to the STP DAT area for the next page, or must indicate no more pages.

JOB *jname* - JTA IMAGE ROLL DAT PAGE NUMBER ERROR

The roll image DAT in the JTA is bad; pages are not numbered consecutively.

JOB *jname* - JTA LENGTH (JTL) *jtl* JXT LENGTH (JXJTL) *jxjtl* MISMATCH

The JTA and JXT disagree on the JTA size.

JOB *jname* - JXT NAME *jxt* MISMATCH JTA NAME *jta*

The JXT and JTA job names disagree. The name given as *jname* is from the index entry.

JOB *jname* - JOB SUCCESSFULLY RECOVERED

JOB *jname* - NO INPUT SDT

No input queue entry exists with a matching job sequence number. This could result if a device were removed from the configuration.

JOB *jname* - REQ SIZE *size* GREATER THAN AVAIL. LWA = *lwa*, FWA = *fwa*

Insufficient memory is available to read part of the JTA or the last block of the user field. FWA and LWA give the first and last available memory addresses.

JOB *jname* - ROLL DAT POINTS TO DOWN DEVICE *ldv*

EQT entry for *ldv* shows device down (EQNA equal to 1).

JOB *jname* - ROLL DAT POINTS TO MISSING DEVICE *ldv*

No EQT entry for device ldv can be found.

JOB *jname* - ROLLOUT APPARENTLY INCOMPLETE

First and last block data/time words disagree.

For the above job-related messages, an additional message is appended at the end of the main message text giving the job sequence number of the job described in the message. This message has the format:

(JSQ=*nnnnn*)

RECOVERY OF THE SYSTEM DIRECTORY 5

5.1 INTRODUCTION

Whenever a Restart or a Deadstart is performed by the operating system, the resident System Directory (SDR) is recovered unless the operator specifies that recovery is not to be performed by means of the *SDR parameter. When an Install is performed, the System Directory is not recovered, and a user job (JSYSDIR) must be run to create the initial System Directory entries.

5.2 SYSTEM DIRECTORY DATASET

A permanent dataset, $SDR, is maintained to contain records specifying system directory datasets to be recovered during a Restart or Deadstart. The Dataset Catalog (DSC) contains an entry for $SDR, which is initialized during an Install. Space is allocated based on the number of SDR entries specified in the system. During a Restart or Deadstart, the dataset is read to rebuild the System Directory. If a failure occurs, a message is issued to the system log, and the operating system initialization is abnormally terminated.

The $SDR dataset is constructed of 512-word blocks. Each block contains eight logical records, with the first word of each block holding the block number relative to the beginning of the $SDR file. The first block in the dataset is a header record containing the maximum number of SDR entries as specified in the last system that recovered the System Directory. The value is updated if the number of entries in the system increases or decreases and recovery is not to be performed. Logical records in the file are accessed by using the formula: (Relative resident SDR entry +1)/8. The quotient gives the block number of the entry within the file, and the remainder gives the logical record number within the block.

Each $SDR record except the header contains the Permanent Dataset Definition Table (PDD) of a dataset entered into the System Directory. When an ACCESS request with the ENTER operand is processed by the Exchange Processor (EXP), the Dataset Name Table (DNT) of the dataset is saved in the resident SDR table. The PDD of the dataset is written to the $SDR dataset. The dataset update is complete before EXP completes processing the request.

5.3 SDRREC SUBROUTINE

SDRREC is executed as a closed subroutine that is called by Z after Recovery of Rolled Jobs (RRJ) is complete but before the system dump is copied. RRJ must be executed first to ensure the integrity of datasets belonging to any jobs being recovered. Any failures during SDR recovery cause the operating system to terminate abnormally.

5.3.1 FILE ALLOCATION

SDR recovery begins with a request to access the $SDR dataset. If no dataset exists, the number of blocks (segments) required to contain the current number of generated resident SDR entries is computed. A request is issued to the Disk Queue Manager to allocate disk space for the dataset. Then a request is made to the Permanent Dataset Manager to SAVE the dataset. Once the operating system initialization is complete, entries can be added to the SDR by ACCESS requests specifying the ENTER parameter.

5.3.2 SDR RECOVERY

If the $SDR dataset exists, each block of the dataset is read and processed until a logical record with a binary zero dataset name is found or until the system-specified number of SDR entries is processed. A Dataset Name Table (DNT) is built for each dataset. The DNT and PDD in the logical record are used to ACCESS the dataset. Then the dataset is entered into the Permanent Dataset Table (PDS). If the dataset access fails, a message is issued to the system log, and the entry is ignored.

5.3.3 NO RECOVERY SPECIFIED

If the operator specifies *SDR in the parameter file to indicate that the System Directory is not to be recovered, a new edition of $SDR is allocated. Once the operating system initialization is complete, entries can be added to the SDR by ACCESS requests specifying the ENTER operand.

5.3.4 CHANGES IN THE NUMBER OF SDR ENTRIES

If System Directory recovery detects that the system-generated number of SDR entries is greater than the value saved in the $SDR header record, the number of blocks required by the system is calculated. If additional blocks are required, write requests are issued until all blocks are allocated. An ADJUST request is issued to the Permanent Dataset Manager to update the DSC for $SDR, and processing continues for SDR recovery.

If System Directory recovery detects that the number of SDR entries specified by the system has decreased, and if no recovery is specified, then the dataset is cleared, and the altered number of SDR entries is recorded in the header record. Once the operating system initialization is complete, entries can be added to the SDR by ACCESS requests specifying the ENTER parameter.

If the number of SDR entries specified by the system has decreased and recovery is to be performed, a message is issued to the system log, and initialization is abnormally terminated.

5.4 MESSAGES DURING SDR RECOVERY

Messages issued during SDR recovery processing are stacked and issued to the system log at the end of Startup. Message texts and interpretations are described below.

5.4.1 MESSAGES ISSUED BY SDR RECOVERY

$SDR, FILE ACCESS FAILED *xxxx*

xxxx is the Permanent Dataset Manager status. The initial ACCESS of the $SDR dataset failed as indicated by the PDM status.

$SDR, FILE ADJUST FAILED *xxxx*

xxxx is the Permanent Dataset Manager status. After an increase in the $SDR dataset size, the ADJUST request failed as indicated by the PDM status.

dataset name FILE ACCESS FAILED *xxxx*

dataset name is the SDR entry dataset name, and *xxxx* is the Permanent Dataset Manager status. The ACCESS request of the specified dataset failed as indicated by the PDM status.

NUMBER OF RESIDENT SDR ENTRIES DECREASED

SDR recovery detected that the number of resident SDR entries in the system is less than the number saved in the dataset, and recovery is requested.

Part 4

SYSTEM MODIFICATION AND MAINTENANCE

CONTENTS
PART 4 SYSTEM MODIFICATION AND MAINTENANCE

1. SYSTEM CODING CONVENTIONS 1-1
   1.1 INTRODUCTION. 1-1
   1.2 SOURCE LANGUAGE 1-1
      1.2.1 CAL source statement format. 1-1
      1.2.2 Comment line 1-2
      1.2.3 Comment field documentation 1-3
   1.3 PROGRAM ORGANIZATION. 1-3
      1.3.1 IDENT statement. 1-3
      1.3.2 ABS statement. 1-4
      1.3.3 BASE statement 1-4
      1.3.4 TITLE and SUBTITLE 1-4
      1.3.5 System definitions 1-5
      1.3.6 Preamble 1-6
      1.3.7 Local definitions. 1-8
      1.3.8 Body 1-9
      1.3.9 END statement. 1-14
   1.4 COMMON DECKS. 1-15
2. SYSTEM MACROS AND OPDEFS 2-1
   2.1 INTRODUCTION. 2-1
      2.1.1 General rules for macros 2-1
   2.2 TABLE MANAGEMENT MACROS AND OPDEFS. 2-2
      2.2.1 TABLE macro. 2-2
      2.2.2 FIELD macro. 2-3
      2.2.3 BUILD macro. 2-4
      2.2.4 ERDEF macro. 2-8
      2.2.5 Partial-word opdefs. 2-8
   2.3 DIVIDE OPDEF. 2-19
3. ADDING A TASK. 3-1
   3.1 TASK ID 3-1
   3.2 INTER-TASK COMMUNICATION. 3-1
   3.3 TASK I/O. 3-2

3.4 TASK SUSPENSION 3-2
3.5 TASK CREATION 3-3
3.6 TASK EXECUTION 3-3

4. SYSTEM DEBUG COMMANDS 4.1-1

4.1 PROGRAM CONTROL COMMANDS 4.1-1
4.1.1 ASSIGN - Assign default job SDT or task number . . 4.1-2
4.1.2 BREAKPOINT - Set program breakpoint 4.1-2
4.1.3 REMOVE - Remove program breakpoint 4.1-3
4.1.4 MODE - Specify default debug mode 4.1-4
4.1.5 RUN - Run job or task 4.1-5
4.1.6 STOP - Stop job or task 4.1-5
4.1.7 INITIATE - Initiate task 4.1-6

4.2 REGISTER AND MEMORY MODIFICATION 4.2-1

4.3 DEBUG DISPLAY COMMANDS 4.3-1
4.3.1 General form of debug displays 4.3-1
4.3.2 Debug display command 4.3-1
4.3.3 Display roll command 4.3-2
4.3.4 DISPLAY - Display definition 4.3-2
4.3.5 ALTER - Alter display definition 4.3-5
4.3.6 DEBUG - Debug display directory 4.3-5
4.3.7 Examples of displays 4.3-6

5. UNB UTILITY PROGRAM 5-1

5.1 INTRODUCTION 5-1
5.2 UNB CONTROL STATEMENT 5-1
5.3 RESTRICTIONS 5-2

6. CRAY-1 SIMULATOR (CSIM) 6-1
6.1 INTRODUCTION 6-1
6.1.1 Simulated configuration 6-2
6.1.2 CRAY-1 CPU simulator 6-2.1
6.1.3 DCU-2 controller 6-2.1
6.1.4 Front-end processor 6-2.1
6.1.5 Bipolar memory 6-2.1

6.1.6 Disk storage. 6-2.2
6.2 CSIM EXECUTION. 6-4
6.2.1 CRAY-OS simulation. 6-4
6.2.2 Checkpoints of the simulated system 6-5
6.2.3 Input of jobs and data. 6-5
6.3 CSIM CONTROL STATEMENT. 6-5
6.3.1 SIMABORT control statement. 6-7
6.4 GENERAL FORM AND SYNTAX OF CSIM DIRECTIVES. 6-7
6.4.1 General form of directives. 6-8
6.4.2 Directive parameters. 6-9
6.4.3 Macros. 6-10
6.5 CSIM DIRECTIVES 6-12
6.5.1 DS - Simulated deadstart. 6-12
6.5.2 CPT - Checkpoint. 6-12
6.5.3 RST - Restart 6-12
6.5.4 EXIT - Simulator error exit 6-13
6.5.5 STOP - Stop instruction interpretation. 6-13
6.5.6 START - Begin instruction interpretation. 6-14
6.5.7 MEM - Enter value into simulated memory 6-15
6.5.8 TRACE, TRACEJP, TRACEXP - Define trace conditions . 6-18
6.5.9 SNAPM, SNAPR, SNAPX - Define snap conditions. . . . 6-21
6.5.10 TRAP - Define trap conditions 6-24
6.6 SIMULATED OPERATOR STATION COMMANDS 6-27
6.6.1 Operator command format 6-27
6.6.2 Channel control directive 6-27
6.6.3 Operator commands delay count 6-28
6.7 SAMPLE CSIM JOB 6-29
7. EXTRACT. 7-1
7.1 INTRODUCTION. 7-1
7.2 EXTRACT STATEMENT 7-1
7.3 DIRECTIVES. 7-1
7.3.1 General format of directives. 7-2
7.3.2 SELECT directive. 7-3
7.3.3 INPUT directive 7-7
7.3.4 OUTPUT directive. 7-7

7.3.5 LINES directive 7-7
7.3.6 FLUSH directive 7-8
7.3.7 NOHEADER directive. 7-8
7.3.8 DUMP directive. 7-8
7.3.9 RAWDUMP directive 7-8
7.3.10 LEFT8 and RIGHT8 directives 7-9
7.3.11 END directive 7-9
7.4 DIRECTIVE SETS. 7-10
7.5 ACCOUNTING RECORD FEATURE 7-10
7.6 EXAMPLES. 7-11
8. SPECIAL CONTROL STATEMENT PARAMETERS 8-1
8.1 INTRODUCTION. 8-1
8.2 BREAKPOINT PARAMETER. 8-1
8.3 SDR ENTRY PARAMETER 8-1
9. FDUMP 9-1
9.1 INTRODUCTION. 9-1
9.2 FDUMP CONTROL STATEMENT 9-1
9.3 FDUMP DIRECTIVES. 9-2
9.3.1 Class 1 directives. 9-2
9.3.2 Class 2 directives. 9-13
9.4 SYSTEM DUMP FORMAT. 9-16
10. SYSTEM STATISTICS (STATS). 10-1
10.1 INTRODUCTION. 10-1
10.2 CONTROL STATEMENT 10-1
10.3 DATASET REQUIREMENTS. 10-2
10.4 EXAMPLES. 10-3

FIGURES
PART 4 SYSTEM MODIFICATION & MAINTENANCE

3-1	CMCC structure	3-2
4.3-1	Debug display format	4.3-1
5-2	Unblocked binary format	5-1
6-1	Relationship of CSIM to CRAY-OS	6-1
6-2	Simulated CRAY-1 configuration	6-2
6-3	Memory assignment of CRAY-1.	6-3
7-1	Example of EXTRACT report.	7-12
7-2	Listing of all subtypes of SPM records	7-13
9-1	Format of a Startup-created dump dataset	9-16
9-2	Format of a dump header	9-17
9-3	Format of a Memory Descriptor Word Pair (MDW)	9-18
9-4	Format of a Dump Control Word (DCW)	9-19

TABLES
PART 4 SYSTEM MODIFICATION & MAINTENANCE

4.1-1 Debug program control command summary 4.1-1
4.2-1 Location-related characteristics 4.2-2
4.3-1 Special keys for controlling debug displays 4.3-2
4.3-2 Display mode dependencies 4.3-4

SYSTEM CODING CONVENTIONS 1

1.1 INTRODUCTION

This section provides specifications for formatting and annotating code in source language programs for the CRAY-OS operating system and any new product set members.

Adherence to these specifications promotes:

- Uniform format and appearance of the listings. The system programmer's job is easier when all of the program elements such as preambles, macros, common deck calls, code body, and so on, are in the same relationship to each other.
- Complete documentation of a given program. All of the detail needed to understand a section of code is available in the correct form. These specifications attempt to define the desired level of detail and proper format.
- Orderly organization. Well organized code reflects an orderliness of thought. The code is easier to write, easier to read, and easier to modify.
- Good annotation of code aids in training new personnel and reduces the risk of degrading the system by later modifications.

1.2 SOURCE LANGUAGE

These coding specifications assume that the programmer is coding in the CRAY Assembler Language (CAL).

1.2.1 CAL SOURCE STATEMENT FORMAT

Although CAL source language statements are free-form, adoption of a standard for the location of fields greatly enhances the readability of the listings.

Column	Contents
1	Asterisk indicating comment card
1-8	Location field entry, left adjusted
9	Blank

Column	Contents
10-18	Result field entry, left adjusted
19	Blank
20-33	Operand field entry, left adjusted
34	Blank
35-72	Comments

Field sizes can be increased to accommodate cases where statement elements cause the standard field size to be exceeded. Remember that a blank must always separate adjacent fields.

1.2.2 COMMENT LINE

A comment line is signified by an asterisk in column one.

Columns two through four may also contain asterisks to delineate a heirarchy of documentation for use by a document extraction program.

The convention here is:

- `*` Comment line; it is a part of formal documentation if it is delimited by lines containing two or three asterisks followed by a blank column.
- `**` First occurrence indicates that internal documentation follows; next occurrence indicates end of formal documentation.
- `***` First occurrence indicates that external documentation follows; next occurrence indicates end of formal documentation.

External documentation consists of comments directed at the general user.

Internal documentation consists of comments directed at the system programmer and describes the internal characteristics of a program, such as entry/exit conditions, timing considerations, etc.

1.2.3 COMMENT FIELD DOCUMENTATION

The programmer may define the purpose of an instruction by providing comments starting with column 35 of the line. Use comments as needed to augment the instructions but do not use the comment field to reiterate an instruction function that is obvious from the instruction, itself.

A poor example:

Location	Result	Operand	Comment
1	10	20	35
	S3	5	(S3)=5

A comment on every line is unnecessary and reduces the readability of the listing. The description of the body of the program gives additional notes for using comments.

1.3 PROGRAM ORGANIZATION

Each program should be composed of the following parts:

- IDENT statement
- ABS statement (if absolute)
- BASE statement
- TITLE and SUBTITLE statements
- System definitions
- Preamble
- Local definitions
- Body
- END statement

1.3.1 IDENT STATEMENT

The IDENT statement provides the name of the program. Conventions are:

- A routine name is 3 to 8 characters
- A routine name should be a mnemonic or acronym pertinent to the function of the routine.

1.3.2 ABS STATEMENT

An ABS statement immediately follows the IDENT statement and declares the program as absolute rather than relocatable.

1.3.3 BASE STATEMENT

A BASE statement customarily follows the ABS statement and defines the mode for assembly of numeric constants as octal or decimal. Decimal is the CRAY-1 CAL default mode and is the recommended mode for all new code. However, a BASE O statement is required if the code is to be assembled under both CRAY-1 CAL and Station CAL. In this case, the BASE statement must precede the IDENT since Station CAL does not support the BASE pseudo.

1.3.4 TITLE AND SUBTITLE STATEMENTS

The proper use of TITLE and SUBTITLE statements and other list control pseudo instructions adds immeasurably to the readability and clarity of a listing.

The first TITLE statement sets the main title, which appears on the left half of the first header line of each page of the listing. The content of this main title should indicate the major program to which the code belongs.

Example of TITLE statement:

Location	Result	Operand	Comment
1	10	20	35
	TITLE	'SYSTEMΔTASKΔPROCESSOR'	

(Δ indicates a space)

A SUBTITLE statement defines the information appearing on the second line of each page of the listing. The content should indicate the subprogram, function, subroutine, or code sequence to which the code belongs.

Example of SUBTITLE statement:

Location	Result	Operand	Comment
1	10	20	35
	SUBTITLE	'USERΔEXCHANGEΔPROCESSORΔTASKΔ-ΔOPENΔDATASET'	

1.3.5 SYSTEM DEFINITIONS

Tables

- All tables are defined in a common place. This will eventually be a set of common decks for the tables.
- Symbols associated with table attributes will adhere to the following conventions

Symbol	Attribute
LH@name	Length of table header
LE@name	Length of table entry
B@name	Beginning address of table
SZ@name	Size of table
NE@name	Number of entries in table

The name portion of a symbol represents the name of the table being described. This name should not exceed 3 characters.

- Standard system macros are provided for table definition, reference and manipulation (section 2).
- For table references:

  (A1) Base table (entry) address

  (exp) Word number, symbolic

Fields

- All fields within a table are defined in a common place. This is currently in each program but will eventually be a common deck.
- Symbols associated with fields will adhere to the following conventions

Symbol	Attribute
W@name	Relative word in table or table entry in which field resides.
S@name	Leftmost bit number of the field in the word.
N@name	Field size in number of bits.

  The name portion of a symbol represents the name of the field being described. It is 3 to 6 characters where the first 2 characters mnemonically identify the table containing the field.
- Standard system macros and opdefs are provided for field definition, reference, and manipulation (section 2).

1.3.6 PREAMBLE

A preamble is a set of comment lines that provides descriptive information about a sequence of code. A preamble must precede each task, routine, or subroutine.

The function of a preamble is to identify a unit of code and to provide a general understanding of how to use the code, what the code does, and how it performs its function.

A preamble has the following general structure:

Column 72

```
**************************************************************...******
**
*NAME  name
       (The name identifies the code being described.)
*PURPOSE
*      A statement of the purpose of the section of code may be given here.
*ENTER
*      A list of the entry conditions and parameters required is given
*      here.  Indicate registers, tables, etc.
*EXIT
*      Output conditions that exist after code execution are noted here.
*      Include any parameters returned by the code.
*METHOD, or STEPCHART, or PROGRAM FLOW

*      The text supplied with these categories explains functions
*      performed by the code as well as the methods used to perform
*      them.  A programmer must use at least one of these categories.
**
**************************************************************...******
```

Other categories can be included if the programmer feels additional information is needed to understand the code. Examples are:

ABNORMAL EXITS (Any exits that can be taken during code execution are noted here.)

CHANGE LOG (Gives date and description of change)

TIMING CONSTRAINTS

RESTRICTIONS

COMMAND, STATEMENT, OR FUNCTION CODE PROCESSED

CALLING SEQUENCE

DATA READ OR MODIFIED

MESSAGES

The preamble begins and ends with a line of all asterisks; all text in the preamble begins in column 10.

The topics should appear in the order specified.

Follow the preamble with a SPACE 4 and SUBTITLE pseudo instruction (if appropriate).

1.3.7 LOCAL DEFINITIONS

Place any definitions of local parameter (table) areas here.

Observe the following rules for macros and opdefs:

- Where macros and opdefs are defined by the operating system, do not redefine them within a program.
- Use table definition macros for local parameter areas.
- Use macros for code generation sparingly to aid readability.
- Structure macros to make use of data already in registers.
- Extensive use of macros renders code difficult to read.
- Avoid using locally defined macros, their use can seldom be justified.

1.3.8 BODY

The body consists of the program, itself. Recommended coding practices deal with a variety of CAL instructions.

General programming practices

- Keep code as straightforward as possible.
- Self modifying code is strictly prohibited.
- Avoid optimizing code by interleaving unrelated instructions. While this saves a few clock periods, it reduces readability of the code. In general, such optimization does not noticeably improve system performance; good design on a more global scope is much more important.
- When writing code that processes parameters passed from a user program, carefully validate the parameters to protect the system. This rule also holds for the interface between two areas of the system. Remember that self-validation of the system is never bad.
- Code is incomplete if end cases and abnormal exits are not handled fully. Consider them in the design phase, before beginning coding.
- Be sure that fields are large enough to accommodate the maximum possible configuration.
- Use ,0 rather than ,A0 as a suffix for load and store instructions. The use of A0 makes the instruction appear to be indexed.

Word and parcel definitions

- Avoid using P. and W. prefixes if they are redundant. P. is always redundant as a prefix to the special element * (e.g., as in J P.*). A W. prefix to a symbol defined with word address attribute is equally redundant (e.g., as in "W.CONLOC,0 S1") where CONLOC is defined in the location field of a CON pseudo instruction.

- Use the = pseudo instruction to define parcel addresses in code. Do not use the location field of an instruction.

RIGHT:

Location	Result	Operand	
1	10	20	35
LOCSYM	=	*	
	S1	S1&S2	

WRONG:

Location	Result	Operand	
1	10	20	35
LOCSYM	S1	S1&S2	

Bit Counts

- Bit counts should always be specified as decimal values.

RIGHT:

Location	Result	Operand	
1	10	20	35
N@FIELD	=	D'24	

WRONG:

Location	Result	Operand	
1	10	20	35
N@FIELD	=	0'30	

- Shift and mask counts should be expressed symbolically; 64 is the only exception.

EXAMPLE:

Location	Result	Operand	
1	10	20	35
	S2	>N@FIELD	
	S3	S3>D'64-S@FIELD-N@FIELD	

Note: This usage will diminish as the use of GET and PUT macros grows.

Branch Instructions

- Do not use relative address increments in an address expression for a branch instruction (e.g., TAG+1 or *+3). Remember that the destination instruction could move on the next reassembly. The only exception is branching to a table of branch instructions.
- Express comments on a branch instruction as an if condition, sometimes also adding the action taken when the branch is taken.

Example:

Location	Result	Operand	Comment
1	10	20	35
	JSN	TAG	IF TABLE FULL, TAKE ERROR EXIT

Data Generation

- Limit the VWD pseudo instruction to no more fields than required to generate one word (64 bits of data).

RIGHT:

Location	Result	Operand	Comment
1	10	20	35
	VWD	16/P1,48/P2	

WRONG:

Location	Result	Operand	Comment
1	10	20	35
	VWD	64/TAG1,64/TAG2	

- Specify only one piece of data per line of code. Placing several items of data after a DATA pseudo instruction makes the listing difficult to read.
- Specify numeric data in its natural form, that is, readable and understandable. If conversion considerations make this impossible, give the natural form in the comments field.
- Specify character data in one of the allowable character data forms, not as octal values.

Subroutine Tag Conventions

- A subroutine name consists of a 3- to 8-character mnemonic.
- A jump tag consists of the subroutine name or a subset of the subroutine name with 1 to 999 appended, in sequential order.

Register Usage

- Any and all registers are available in user code.
- With the exception of register B0 which is saved on all interchanges, do not use B, T, and V registers in code for STP.
- Refer to A, S, and V registers with their numeric designators (i.e., A0-A7, S0-S7, V0-V7); refer to B and T registers (other than B0) with symbolic designators (i.e., B.sym or T.sym). Suggested symbols are simply two letters as in the case of CFT (i.e., AA-AZ, BA-BZ, CA-CL), or a subset of the subroutine name and a 2-character mnemonic as in the case of CAL.

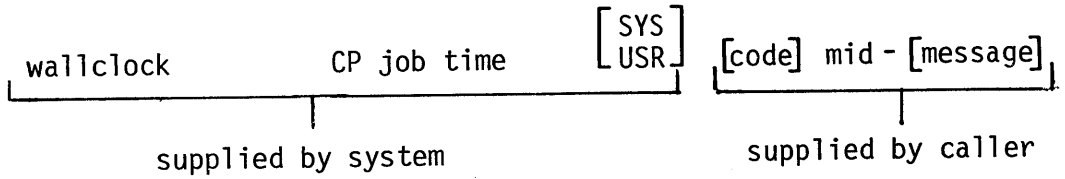
Comments

The code body must include comments describing the code in enough detail to enable a reader to interpret the code.

- Follow individual SUBTITLE statements with comment lines or a preamble that tell what the step or phase is to do.
- Precede each code sequence (10-40 lines) that performs a specific operation with one or more comment lines. These comment cards should convey a general understanding of the code to the reader and should be on a higher level than the comments on the instruction lines. These comment lines can immensely improve the comprehensibility of the code, especially if they are not too terse or cryptic in style.
- Clearly explain logical branch alternatives.
- Avoid comments that exactly parallel the code. Try to generate comments that augment the code.

Logfile Message Formats

Messages for both the system and user logfile have the following format:



code Two characters denoting source of message; required for all CRI-supplied software:

Code	CRI program
LD	Loader
FT	FORTRAN
UT	Utilities

Code	CRI Program
LG	Logical I/O
CS	Control Statement Processor
IO	Physical I/O
PD	Permanent Dataset
ST	Station
CA	CAL

mid — Three-character message identifier assigned by each CRI software area.

message — One- to 80-character message. Avoid overly cryptic messages. Remember that error codes alone are not very informative.

Example of message as it appears on listing:

```
15.26.32  00032.276  USR  UT113 - EOD  ENCOUNTERED ON DN - XYZ.
```

1.3.9 END STATEMENT

The END pseudo instruction is the last line of a program.

1.4 COMMON DECKS

Programs in the operating system access and share macros, tables, symbols, and common code through the use of UPDATE common decks. These decks are used:

- As common field definitions for tables, symbols, and parameters not provided in operating system text.
- To hold locally used field definitions for tables, symbols, and parameters.
- To hold macro and opdef definitions.
- To hold code of common use for more than one part of the operating system.

User decks and common decks are specially identified as such to UPDATE and are maintained on an UPDATE program library. A reference to a common deck is in the form of an UPDATE call directive in the user program deck. The user must run the UPDATE program to obtain a complete deck, that is, a deck composed of the user deck and the common deck.

Directives accompanying the UPDATE call statement tell UPDATE to extract the user programs from the program library, modifying them if desired. UPDATE then searches the user's deck for common deck calls. If it finds a common deck call, it replaces the call with a copy of the common deck and writes the expanded deck onto the compile dataset to be submitted to the language processor (e.g., CFT or CAL).

SYSTEM MACROS AND OPDEFS 2

2.1 INTRODUCTION

The operating system text for STP and EXEC contains a set of system macros and opdefs that perform common system functions. These macros are accessible to CAL-language programs when the S=*systemtext* option on the CAL control statement names the proper system text dataset.

Programmers should take advantage of these standard tools because their consistent use promotes the integrity of the operating system, especially with regard to system table management.

The following groups of macros and opdefs are currently available:

- Table management instructions
- Divide opdef instruction
- Flaw macros (described in Part 2, section 2.1.2)

2.1.1 GENERAL RULES FOR MACROS

Macros are available in the CAL assembler language and are subject to the rules defined in the CAL reference manual. Macros are written in the following general format:

Location	Result	Operand
loc	name	$a_1,a_2,\ldots,a_j,f_1=b_1,f_2=b_2,\ldots,f_k=b_k$

loc — Location field argument; certain macros will require an entry in this field. The assembler assigns loc a word address.

name — Name of macro as given in system text

a_i — Actual argument string corresponding to positional parameter in prototype. Two consecutive commas indicate a null string.

$f_j=b_j$ — Keyword and actual argument; these entries can be in any order. A space or comma following the equal sign indicates a null string.

A parameter shown in all uppercase letters must be coded literally as shown. A parameter presented in lowercase letters must be supplied with a value, symbol, expression, or register designator as indicated in the text following the format.

A macro can be coded through column 72 of a line. It is continued on the next line by placing a comma in column one of the next line and resuming the parameter list in column 2.

2.2 TABLE MANAGEMENT MACROS AND OPDEFS

The system macros and opdefs described in this section aid the programmer in building tables, describing fields within tables, and storing or fetching field values in tables. The macros include:

TABLE	Defines table attributes.
FIELD	Describes a field within a table.
BUILD	Constructs table at assembly time.

Opdefs include:

GET	Fetches the contents of a field.
SGET	Fetches the contents of a field (short form).
PUT	Stores the contents of a field.
SPUT	Stores the contents of a field (short form).
SET	Packs a field value into a word along with other fields.
GETDA	Obtains the STP relative address of the first DAT page.
GETNDA	Obtains the STP relative address of the next DAT page.

2.2.1 TABLE MACRO

The TABLE macro describes a table in terms of special labels for the length of its header, length of each entry, and number of entries.

Format:

Location	Result	Operand
name	TABLE	LH=lh,LE=le,NE=ne,{L=lt / SZ=st}
LH@name	=	lh
LE@name	=	le
NE@name	=	ne
{L@name / SZ@name}	=	{lt / st}

name	The symbol in the location field specifies the name of the table and should be a maximum of 3 characters.	
LH=lh	Length of table header in words.	Attributes can be in any order
LE=le	Length of entry in words.	
NE=ne	Number of entries in the table.	
L=lt or SZ=st	Length (size) of the table in words	

A programmer can define a symbol equal to the size of a table containing a header and multiple entries through the following instruction:

Location	Result	Operand
SZ@name	=	LH@name+LE@name*NE@name

Example:

Location	Result	Operand
DRT	TABLE	LH=3,LE=67,NE=2,SZ=LH@DRT+LE@DRT*NE@DRT

2.2.2 FIELD MACRO

A programmer uses the FIELD macro to set up the special labels for fields in tables. These labels can then be used by GET, SGET, PUT, SPUT, and SET macros.

Format:

Location	Result	Operand
name	FIELD	word,start,number
W@name	=	word
S@name	=	start
N@name	=	number

name — The name of the field being described. It can be 3 to 6 characters where the first 2 or 3 characters mnemonically identify the table containing the field.

word — Relative word in table or table entry in which the field resides. * is allowed to suppress the definition of W@name.

start — Starting (leftmost) bit number of the field in the word. Bit 0 is the sign bit. * is allowed to suppress the S@name and N@name definitions; number should be omitted.

number — Field size in number of bits. This field should not be left null (0). * is not allowed here.

Example:

Location	Result	Operand
DNBSZ	FIELD	6,0,24
DNDAT	FIELD	1,40,24
DNBFZ	FIELD	3,12,15
JTUSR	FIELD	16,*
JTXYA	FIELD	*,40,24

2.2.3 BUILD MACRO

The BUILD macro is used with the TABLE and FIELD macros to construct a table at assembly time. The table is defined in terms of the symbols defined by TABLE and FIELD macros. The BUILD macro eliminates the need for coding VWD statements to set initial values.

Format:

Location	Result	Operand
loc	BUILD	pfx,length,(list)

loc — Optional location symbol. If present, it is defined as the location of word 0 of the table being built and has an attribute of word. BUILD always forces the location counter to a word boundary before beginning to construct the table.

pfx — A 2- or 3-character prefix that identifies the field definition for the table being built. This is not necessarily the table name as specified on the TABLE macro but is the prefix word used in the FIELD macro (e.g., DP for DSP or EQ for EQT).

length — Designates length of table. Normally, this should be L, LH, LE, or SZ. BUILD concatenates the length designator with the @ character and pfx to produce a symbol that defines the length of the table designated by the TABLE macro location field.

If the length of the table cannot be given by concatenating the L, LH, LE, or SZ characters with @pfx, it must be given in the format (lthsym$_{\wedge}$) with the parentheses and blank mandatory.

NOTE

In this section, ^ is used to denote a blank.

(list) — Field names and values to be entered into the field. The list must be enclosed in parentheses unless only one field name and value is specified or no field is specified, in which case the table is initialized with all zeros. A field name may be any characters for which a FIELD macro has been assembled, without the 2- or 3-character prefix. Fields can be in any order. The list must not contain the \ character. Entries in the list have the form:

fieldname=value

where *value* is the data to be assembled into the field and may contain any characters (except \) legal in the value field of a VWD instruction.

Fields specified in the BUILD macro must neither overlap nor be repeated although these conditions are not diagnosed.

Examples:

1. This general-case example defines a table and fields and constructs the table.

   (a) The following defines a table named TAB; all field definitions will be prefixed by TAB.

Location	Result	Operand
TAB	TABLE	LE=2Ø,NE=3,LH=2

   (b) The following define some fields for a table entry.

Location	Result	Operand
TABF1	FIELD	Ø,Ø,64
TABF2	FIELD	2,1,4
TABF3	FIELD	3,6,12

   (c) The following define some fields for the table header.

Location	Result	Operand
TABHF1	FIELD	Ø,Ø,16
TABHF2	FIELD	1,Ø,64

   (d) The following constructs the table during assembly.

Location	Result	Operand	
B@TAB	BUILD	TAB,LH,(HF1='TB'L,HF2='HDR^2'L)	Construct header.
	BUILD	TAB,LE,(F1='TABF1'L,F2=0'37)	Build one entry.
	DUP	NE@TAB-1,1	
	BUILD	TAB,LE	Build remaining entries for all fields initialize to zero.

2. This example builds an EQT entry. Refer to Part 5 of this manual for field names.

Note the use of parentheses and blank character in the length entry field. This is because the length of an EQT entry or header is not defined in terms of the same prefix as the field names. The expansion of the macro yields the following line:

Location	Result	Operand
%TL	SET	LE@EQT^@EQ

The blank before @EQ terminates the assembler scan of the line. If the blank is ommitted, the expansion is:

Location	Result	Operand
%TL	SET	LE@EQT@EQ

which produces an assembly error. Omitting the parentheses produces the same expansion.

(a) The following builds the EQT header.

Location	Result	Operand
B@EQT	BUILD	EQ,(LH@EQT^),(NE=I@NDD819,NUA=Ø,TN='EQT'L)

(b) The following builds the entry for the master device.

Location	Result	Operand
EQTØØ	BUILD	EQ,(LE@EQT^),(LDV='DD-19-2Ø'L,MSD=1,CH1=2,UN=Ø,------ ,----------DRT=DRTØØ,CPD=NCY819,TPC=NTK819,AU=NBL819,LNK=EQTØ4)

(c) The following builds an entry for a non-master device.

Location	Result	Operand
EQTØ1	BUILD	EQ,(LE@EQT^),(LDV='DD-19-3Ø'L,CH1=3,UN=Ø,DRT=DRTØ1,-- ,----------CPD=NCY819,TPC=NTK819,AU=NBL819,LNK=EQTØ5)

2.2.4 ERDEF MACRO

The ERDEF macro generates entries for the error processing table in EXP at assembly time. The entries are used during abort processing.

Location	Result	Operand
	ERDEF	*addr,fatal,class*[,DN=YES][,REPR=NO]

addr Message address for this error

fatal Bit number in JTFEFW if this is a fatal error; otherwise it is 0.

class Major error class. To interpret the value of this table entry, shift a rightmost 1 bit to the left as many times as specified in the table entry. If the table entry is 2, it would be 100_2 or 4_{10}.

DN YES, if a dataset name is to be included with the message; otherwise it is omitted.

REPR NO, if the error is not reprievable; otherwise it is omitted.

2.2.5 PARTIAL-WORD OPDEFS

Partial-word opdefs provide for storing or retrieving a field value in a table. GET and PUT are sufficient; however, SGET, SPUT, and SET are provided for object-time code efficiency. These opdefs assume that the field to be fetched, stored, or packed is in a word that is already in a register as a result of an earlier GET call. Additionally, SPUT assumes that the GET call was for the same field to be stored and that the mask for the field is in a register ready for use by SPUT.

NOTE

When a GET is for a field occupying a full word, CAL does not generate a mask. In this case, SPUT cannot be used after the GET.

PUT

The PUT opdef provides a means of conveniently storing a field entry in a table.

Format:

Location	Result	Operand
symbol	PUT,Sa	Sb&Sc,name,Ad

name — Name of field being stored

Sa — S register containing value to be stored in field; unchanged by PUT.

Sb — S register to be used by PUT to hold full word. S0 is legal.

Sc — S register to be used by PUT to hold bit mask of field. Sa, Sb, and Sc must be unique registers.

Ad — A register containing pointer to base of table or table entry; the contents of this register are unchanged by PUT.

Example:

Location	Result	Operand	Comment
1	10	20	35
TAG	PUT,S2	S3&S4,SDDAT,A1	
TAG	$PUT	SDDAT,2,3,4,1	
TAG	S.3	W@SDDAT,A.1	
	$SET	SDDAT,2,3,4	
	S.4	<N@SDDAT	
	S.3	S.2!S.3&S.4	
	W@SDDAT,A.1	S.3	

GET

The GET opdef provides a means of conveniently fetching a field entry from a table. The field specified as "name" is transferred right-justified into Sa.

Format:

Location	Result	Operand
symbol	GET,Sa	Sb&Sc,name,Ad

name — Name of field being fetched

Sa — S register to receive value being fetched from field

Sb — S register to be used by GET to hold full word from which field is extracted. Sb can be the same as Sa; however an SPUT opdef cannot follow since Sb no longer contains the full word.

Sc — S register to be used by GET to hold bit mask of field. Sc cannot be the same as Sb. If Sc is the same as Sa, then SPUT cannot be used in the following code. The mask will no longer be in Sc.

Ad — A register containing pointer to base of table or table entry; The contents of this register are unchanged by GET.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GET,S2	S3&S4,SDDAT,A3	
	$GET	SDDAT,2,3,4,3	
	S.3	W@SDDAT,A.3	
	$SGET	SDDAT,2,3,4	
	S.4	<N@SDDAT	
	S.2	S.3&S.4	

SPUT

SPUT resembles PUT but can be used after a GET or SGET for the same field in order to generate less object-time code.

Format:

Location	Result	Operand
symbol	SPUT,Sa	Sb&Sc,name,Ad

name — Name of field being stored.

Sa — S register containing value to be stored in field; contents of this register are not changed by SPUT.

Sb — S register assumed to contain full word in which field is to be stored (output from GET).

Sc — S register assumed to contain bit mask for field (output from GET). Sa, Sb, and Sc must be unique registers.

Ad — A register containing pointer to base of table or table entry; the contents of this register are the same as for the GET and are unchanged by the PUT.

Example:

Location	Result	Operand	Comment
1	10	20	35
	SPUT,S5	S6&S7,JXCJS,A4	
	$SPUT	JXCJS,5,6,7,4	
	S.6	S.5!S.6&S.7	
	W@JXCJS,A.4	S.6	

SGET

When a programmer is working with more than one field from a word, SGET rather than GET can be used for second and subsequent fields to reduce the amount of object code generated. SGET can follow GET, PUT, or SPUT on the same word.

Format:

Location	Result	Operand
symbol	SGET,Sa	Sb&Sc,name

name — Name of field being fetched.

Sa — S register to receive value being fetched from field.

Sb — S register assumed to hold full word from which field is extracted (output from GET, PUT, or SPUT). Sb can be the same as Sa; however, an SPUT opdef cannot follow since Sb no longer contains the full word.

Sc — S register to be used by SGET to hold bit mask of field. Sc cannot be the same as Sb. If Sc is the same as Sa, then SPUT cannot be used in the following code. The mask will no longer be in Sc.

Example:

Location	Result	Operand	Comment
1	10	20	35
	SGET,S0	S3&S4,SDDAT	
	$SGET	SDDAT,0,3,4	
	S.4	<N@SDDAT	
	S.0	S.3&S.4	

SET

The SET opdef can be used to pack several fields into a word. The caller must code a store instruction at the end of a series of SET opdefs. The SET must follow a GET if new information is being packed into portions of an existing word. If the entire word is changing, the register to receive the newly packed information can be zeroed.

Format:

Location	Result	Operand
symbol	SET,Sa	Sb&Sc,name

name — Name of field to be packed.

Sa — S register containing value to be packed into field of Sb

Sb — S register in which word is being packed ready for storage

Sc — S register to be used by SET to hold bit mask for field. Sa, Sb, and Sc must be unique registers.

Example:

Location	Result	Operand	Comment
1	10	20	35
	SET,S3	S6&S7,JXBLO	
	$SET	JXBLO,3,6,7	
	S.7	>N@JXBLO	
	S.7	S.7>S@JXBLO	
	S.3	S.3<D'64-N@JXBLO-S@JXBLO	
	S.6	S.3!S.6&S.7	
	S.3	S.3>D'64-S@JXBLO-N@JXBLO	

GETDA

The GETDA opdef obtains the STP relative address of the first DAT page. The opdef has two formats: a short form if both the DNT and the JTA address are known, and a long form if only the DNT address is known.

Format (long form):

Location	Result	Operand
symbol	GETDA,Aa,Ab	Sc&Sd,Ae,Af

Aa A register to receive the STP relative DAT address; cannot be AØ.

Ab A register to receive the STP relative JTA address if the DAT resides in the JTA; cannot be AØ.

Sc S register to be used by GETDA.

Sd S register to be used by GETDA.

Ae A register to be used by GETDA; cannot be AØ.

Af A register containing STP relative DNT address; cannot be AØ.

CAUTION

AØ is destroyed by this macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETDA,A1,A2	S3&S4,A5,A6	
	S.3	W@DNDAT,A.6	
	A.1	S.3	
	AØ	A.1	
	JAP	GN1	
	SGET,S.4	S.3&S.4,DNJORD	
	A.2	S.4	
	A.5	LE@JXT	
	A.2	A.2*A.5	
	A.5	B@JXT	
	A.2	A.2+A.5	
	A.2	W@JXJTA,A.2	
	A.1	A.2-A.1	
GN1	=	*	

Format (short form):

Location	Result	Operand
symbol	GETDA,Aa	Ab,Ac

Aa A register to receive the STP relative DAT address; cannot be AØ.

Ab A register containing the STP relative DNT address; cannot be AØ.

Ac A register containing the STP relative JTA address; cannot be AØ.

CAUTION

AØ is destroyed by this macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETDA,A1	A2,A3	
	A.1	W@DNDAT,A.2	
	AØ	A.1	
	JAP	GD1	
	A.1	A.3-A.1	
GD1	F	*	

GETNDA

The GETNDA opdef obtains the STP relative address of the next DAT page. The opdef has two formats. A short form if both the current DAT page address and the JTA address are known, and a long form if only the current DAT page address is known.

Format (long form):

Location	Result	Operand
symbol	GETNDA,Aa,Ab	Sc&Sd,Ae,Af

Aa A register to receive the STP relative DAT address; cannot be AØ.

Ab A register to receive the STP relative JTA address if the DAT is in the JTA; cannot be AØ.

Sc S register to be used by GETNDA.

Sd S register to be used by GETNDA.

Ae A register to be used by GETNDA; cannot be AØ.

Af A register containing the STP relative address of the current DAT page; cannot be AØ.

CAUTION

AØ is destroyed by this macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETNDA,A1,A2	S3&S4,A5,A6	
	S.3	W@DADAT,A.6	
	A.1	S.3	
	AØ	A.1	
	JAP	GN1	
	SGET,S.4	S.3&S.4,DAJORD	
	A.2	S.4	
	A.5	LE@JXT	
	A.2	A.2*A.5	
	A.5	B@JXT	
	A.2	A.2+A.5	
	A.1	A.2-A.1	
GN1	=	*	

CAUTION

AØ is destroyed by this macro.

Format (short form):

Location	Result	Operand
symbol	GETNDA,Aa	Ab,Ac

Aa A register to receive the STP relative DAT address; cannot be AØ.

Ab A register containing the STP relative address of the current DAT page; cannot be AØ.

Ac A register containing the STP relative JTA address; cannot be AØ.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETNDA,A1	A2,A3	
	A.1	W@DADAT,A.2	
	AØ	A.1	
	JAP	GD1	
	A.1	A.3-A.1	
GD1	=	*	

Sample code for field management

In the following code sequence, the JXT entry is removed from the MP chain and the JXT entry is added to the end of the available chain.

```
Location  Result        Operand
          A7            B@JXT
*       REMOVE THE JXT ENTRY FROM THE MP CHAIN.
          GET,S4        S6&S7,JXFLO,A4        S4=FLINK(A4)
          S6            #S7&S6                FLINK(A4)=0
          SGET,S3       S6&S7,JXBLO           S3=BLINK(A4)
          A3            S3
          A5            S4
          A3            A7+A3                 A3=BLINK(A4)
          A5            A7+A5                 A5=BLINK(A4)
          PUT,S4        S1&S2,JXFLO,A3        FLINK(A3)=S4
          PUT,S3        S1&S2,JXBLO,A5        FLINK(A5)=S3
*       ADD THE JXT ENTRY TO THE END OF THE AVAILABLE CHAIN.
          S1            Q@JXAVL,0             S1=AVAIL HEAD
          SGET,S3       S1&S2,JXBLO           S3=BLINK(AVAIL HEAD)
          A5            A4-A7
          A3            S3
          S5            A5                    S5=OFFSET OF DELETED ENTRY
          A3            A7+A3                 A3=BLINK(AVAIL HEAD)
          SET,S5        S1&S2,JXBLO           BLINK(AVAIL HEAD)=S5
          Q@JXAVL,0     S1
          PUT,S5        S1&S2,JXFLO,A3        FLINK(A3)=S5
          SET,S3        S6&S7,JXBLO           BLINK(A4)=S3
          S4            J$K                   STATUS(A4)=K
          SPUT,S4       S6&S7,JXSTAT,A4
```

2.3 DIVIDE OPDEF

The divide instruction enhances the instruction repertoire of the CAL assembler by providing a precoded divide routine accessible through a call in machine language syntax.

Format:

Location	Result	Operand
symbol	Si	Sj/FSk
symbol	Si	/HSk
	Sj	Sj*FSi
	Sk	Sk*ISi
	Si	Sj*FSk

The contents of registers Sj and Sk are destroyed.

ADDING A TASK 3

Adding a task to STP affects many areas of the system. This section provides a brief guideline for implementing a task design consistent with system conventions.

3.1 TASK ID

Each task has a symbolic ID and a numeric ID. The symbolic task ID is the label of a word that contains the numeric ID. These IDs are in noncontiguous words.

When defining a new task, the programmer assigns a 5-character symbolic ID to a word (in the form xxxID, where xxx are unique to the task) and stores an integer from 0 to 35_8 in the word. Usually, the integer represents the next available task number.

Example:

Location	Result	Operand	Comment
1	10	20	35
NEWID	BSSZ S6 NEWID	1 10 S6	Task ID word Numeric ID of task

3.2 INTER-TASK COMMUNICATION

Tasks communicate with each other through Communication Modules (CMODs). CMODs are obtained from a memory pool. Creation of a new task may require expansion of the memory pool.

The CMODs are controlled by the Communication Module Chain Control (CMCC). When a new task is added to the system, a CMCC must be created to accommodate communication from all tasks to the new task. Additionally, each existing CMCC must be expanded to facilitate communication between the new task and all previously existing tasks.

The CMCCs are ordered and accessed according to numeric task ID. Therefore, it is essential that changes are properly handled. The CMCCs must be structured as illustrated in figure 3-1.

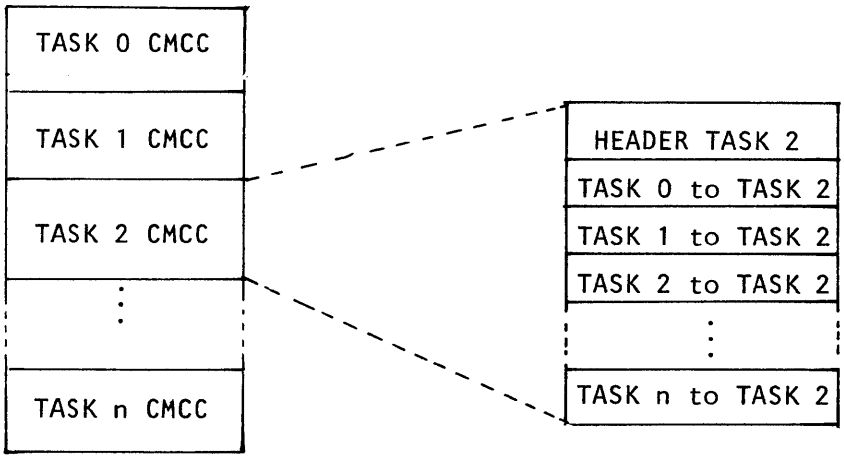


Figure 3-1. CMCC structure

The new task and other STP tasks must use the common communication subroutines (TSKREQ, PUTREQ, GETREQ, PUTREPLY, and GETREPLY) for all inter-task communication. Refer to Part 1, section 3.2 for a detailed description of inter-task communication.

3.3 TASK I/O

Tasks should use TIO for system I/O (refer to section 3.3 of part 1).

3.4 TASK SUSPENSION

Code the task so that it has a single suspend point.

3.5 TASK CREATION

Insert a create task request (CTSK) to EXEC into Task 0 so that the new task can be scheduled for execution. Be sure to use the same task ID for this call as was used in inter-task communication.

The CTSK call also generates the task's entry into STT, sets the task's priority and execution address, and identifies its exchange package. There is no need to expand the STT since it is large enough to accommodate over thirty tasks.

The interaction of the new task with the rest of STP is governed by task priority set on the create task call. The higher the task's priority, the more responsive it is to requests for service. The COS task priorities assure minimum response time to external interrupts. That is, task priorities are generally arranged so that those tasks having the least to do with external interrupts have the lowest priorities. The priority of a new task should be chosen to be consistent with this convention.

3.6 TASK EXECUTION

The new task can execute as a result of implicit or explicit calls. If called implicitly, it is triggered by an interrupt on a pseudo channel. For interrupts to be recognized and correlated with a task, the task must be assigned to the channel through an ARES, FET, or I/O request of EXEC. ARES assignments are usually made during task initialization. The FET and I/O assignments currently are made dynamically by the SCP and DQM tasks. It is unlikely that any new task would make FET and I/O calls since by doing so they could cause serious conflicts in the system.

Task execution is triggered by an explicit call whenever some other task makes an RTSS or RTSK call to EXEC for the task. Such calls should use the symbolic name of the task.

SYSTEM DEBUG COMMANDS 4

The Data General Station provides the system analyst with a set of debug commands and displays in addition to the features described in the DGS Operator's Guide (CRI publication 2240006).

Debug station commands are in three basic formats: program control, register/memory modification, and display.

4.1 PROGRAM CONTROL COMMANDS

The program control command format is the same as the general form for operator commands described in the operator's guide. Table 4.1-1 summarizes debug commands of this type. Commands are described in detail in the following subsections.

Table 4.1-1. Debug program control command summary.

Command	Function
ASSIGN	Designates JSQ number or task number to be used as default for MODE command.
BREAKPOINT	Sets a program stop (breakpoint).
REMOVE	Removes a breakpoint.
INITIATE	Initiates execution of a system task.
MODE	Designates a default program mode that applies for subsequent commands.
RUN	Resumes execution of a job or task that has been suspended.
STOP	Suspends execution of the currently executing job or task.

4.1.1 ASSIGN - ASSIGN DEFAULT JOB SEQUENCE NUMBER OR TASK NUMBER

FUNCTION: Assigns default job sequence number or task number. The default can be overridden by explicitly specifying the job sequence number or task number on the MODE command or on the command for which MODE provides defaults.

FORMAT: ASSIGN,{JOB,*jsq* / TASK,*tn*} ↵

jsq Job sequence number (JSQ) by which job is identified in CRAY-OS system. The JSQ of a job can be obtained through the STATUS command.

tn Number of STP task.

TYPE: Debug

PREREQUISITES: Eclipse must be logged on and must be operator station.

EXAMPLE: ASSIGN,JOB,1 ↵

The job assigned JSQ entry 1 is the default for unqualified references to a job.

4.1.2 BREAKPOINT - SET PROGRAM BREAKPOINT

FUNCTION: Sets a program stop (breakpoint). The program stops when it reaches the breakpoint address. The breakpoint is either cleared so that this breakpoint is no longer available when the program resumes execution or is reset as an option of the command.

FORMAT: BREAKPOINT,*number*,*address*,{*reset*}{,JOB{,*jsq*} / ,TASK} ↵

number Ordinal identifying the breakpoint; 0 through 7.

address Breakpoint address, word or parcel. A parcel address is a word address with a suffix (A, B, C, or D) to indicate the relative parcel within the word or is a parcel pointer (word address times four plus the offse of the parcel within the word, i.e., 0, 1, 2 or 3) wit a P suffix.

reset Breakpoint reset address. When this secondary breakpo address is encountered, the system resets the breakpoi After the breakpoint is reset, program execution resum so that the reset interruption is transparent to the u The reset address has the same rules as *address*.

JOB{,*jsq*} Program area, job or system, in which breakpoint
TASK address is set. If neither JOB nor TASK is specified, the default from the most recent MODE command applies.

Task breakpoints are relative to STP origin, not to the origin of a particular task so that providing a task number is superfluous.

TYPE: Debug

PREREQUISITES: Eclipse is logged on and is operator station.

EXAMPLE: BRE,2,355B,357D,JOB,5 ↵

Breakpoint 2 for the job assigned the JSQ ordinal of 5 causes the job to halt when the instruction at parcel 355B is executed. The breakpoint is reset at parcel address 357D.

4.1.3 REMOVE - REMOVE PROGRAM BREAKPOINT

FUNCTION: Deletes a breakpoint associated with a specific job or with STP.

FORMAT: REMOVE,*number* {,JOB{,*jsq*} / ,TASK} ↵

number Number of breakpoint to be removed.

JOB{,*jsq*} Program area, job or system, in which breakpoint
TASK address is set. If neither JOB nor TASK is specified, the default from the most recent MODE command applies.

Breakpoints for all tasks are relative to STP origin so that providing a task number is superfluous.

TYPE: Debug

REREQUISITES: Eclipse is logged on and is operator station.

EXAMPLE: REM,2,JOB,5 ↵

Breakpoint 2 for the job with JSQ ordinal 5 is cleared.

4.1.4 MODE - SPECIFY DEFAULT DEBUG MODE

FUNCTION: Assigns a default mode to be used for unqualified debug commands. The JSQ ordinal or task number is optional if an assignment has been made by a previous ASSIGN or MODE command. A JSQ ordinal or task number, if specified, becomes the new default value.

FORMAT:

MODE { ,EXECUTIVE
, JOB { ,*jsq* }
, TASK { ,*tn* }
, STATION } ↵

EXECUTIVE Subsequent commands that do not explicitly designate a mode are assumed to apply to the system executive (EXEC).

JOB { ,*jsq* } Subsequent commands that do not explicitly designate a mode are assumed to apply to the job designated in this command, or if *jsq* is omitted, to apply to the job assigned by the ASSIGN command.

TASK { ,*tn* } Subsequent commands that do not explicitly designate a mode are assumed to apply to the task designated in this command, or if *tn* is omitted to apply to the task assigned by the ASSIGN command.

STATION Subsequent commands for which mode is not explicitly specified are assumed to be for the station.

TYPE: Debug

PREREQUISITES: Eclipse must be logged on and must be the operator station.

EXAMPLE:

ASSIGN,JOB,1 ↵	Job 1 is default.
MODE,JOB ↵	Default mode is JOB.
DISPLAY,X,XP ↵	Describes X display.
X. ↵	Displays Job 1's exchange package.
MODE,EXEC ↵	Displays EXEC's exchange package.

4.1.5 RUN - RUN JOB OR TASK

FUNCTION: Initiates or resumes execution of the specified job, or resumes execution of a currently suspended task.

FORMAT:

```
RUN {,JOB {,jsq}} ↵
    {,TASK       }
```

JOB {,*jsq*} Job for which execution is to be resumed. If neither JOB nor TASK is specified, the default from the most recent MODE command applies.

TASK Resumes execution of the currently suspended task.

TYPE: Debug

PREREQUISITES: Eclipse must be logged on and must be the operator station.

4.1.6 STOP - STOP JOB OR TASK

FUNCTION: Suspends execution of the specified job or of the entire system except SCP.

FORMAT:

```
STOP {,JOB{,jsq}} ↵
     {,TASK     }
```

JOB{,*jsq*} Program (job or system) for which execution is to
TASK be suspended. If neither JOB nor TASK is specified, the default from the most recent MODE command applies.

TYPE: Debug

PREREQUISITES: Eclipse must be logged on and must be the operator station.

4.1.7 INITIATE - INITIATE TASK

FUNCTION: Initiates execution of a task.

FORMAT: INITIATE {,*tn*}

tn	Number of STP task. If *tn* is omitted, the default defined by MODE or ASSIGN applies.

TYPE: Debug

PREREQUISITES: Eclipse must be logged on and must be the operator station.

4.2 REGISTER AND MEMORY MODIFICATION

The register and memory-entry command allows the operator to enter octal or ASCII data into a register or into a memory location. The command has the following general format:

location = *data* {,*mode*} ↓

location Specifies the register or memory location into which data is to be entered. *location* is one of the following:

A*r*	A register *r*, where *r* is 0 - 7.
B*rr*	B register *rr*, where *rr* is 0 - 77_8.
P	P register in the exchange package.
S*r*	S register *r*, where *r* is 0 - 7.
T*rr*	T register *rr*, where *rr* is 0 - 77_8.
VL	Vector length in the exchange package.
VM	Vector mask register.
V*ree*	Vector register *r*, where *r* is 0 - 7 and element *ee*, where *ee* is 0 - 77_8.
address	Word address or parcel address. A parcel address is a word address with a suffix (A, B, C, or D) or is a parcel pointer (word address times four plus the offset of the parcel within the word; i.e., 0, 1, 2, or 3) with a P suffix.

data Specifies the data to be entered into the register or memory location. *data* can be in one of the following forms:

octal	Octal constant
address	Word address (0-377777_8) with a parcel designator (A, B, C, or D) appended
'*ascii*'	ASCII text delimited by apostropnes

The *octal* and *address* values are stored right justified, zero filled. ASCII data is stored left justified with null fill. An error message is issued if the specified data would overflow the location. The maximum data width in bits for locations is shown in table 4.2-1.

mode Designates program area of operating system that is to be modified. If the MODE parameter is not specified, the default defined by the most recent MODE command applies.

Mode can be one of the following. Table 4.2-1 designates which modes apply for the different *location* options.

EXECUTIVE Modifies system executive.

JOB {,*jsq*} Modifies job having the JSQ ordinal *jsq*

TASK {,*tn*} Modifies the task having the task number *tn*. For *address*, *tn* is meaningless; addresses are relative to STP origin.

Table 4.2-1. Location-related characteristics

Location	Data size in bits	Mode		
		EXECUTIVE	TASK	JOB
A*r*	24	NO	YES	YES
B*rr*	24	NO	YES†	YES
P	22	NO	YES	YES
S*r*	64	NO	YES	YES
T*rr*	64	NO	NO	YES
VL	7	NO	YES	YES
VM	64	NO	NO	YES
V*ree*	64	NO	NO	YES
word address	64	YES	YES	YES
parcel address	16	YES	YES	YES

† B_{00} only

4.3 DEBUG DISPLAY COMMANDS

4.3.1 GENERAL FORM OF DEBUG DISPLAYS

Figure 4.3-1 illustrates the debug display format. Note that the header and command/response areas are the same as described in the DGS Operator's Guide (CRI publication 2240006). However, for debug displays, the display area is divided into a 40-character left display area and a 40-character right display area.

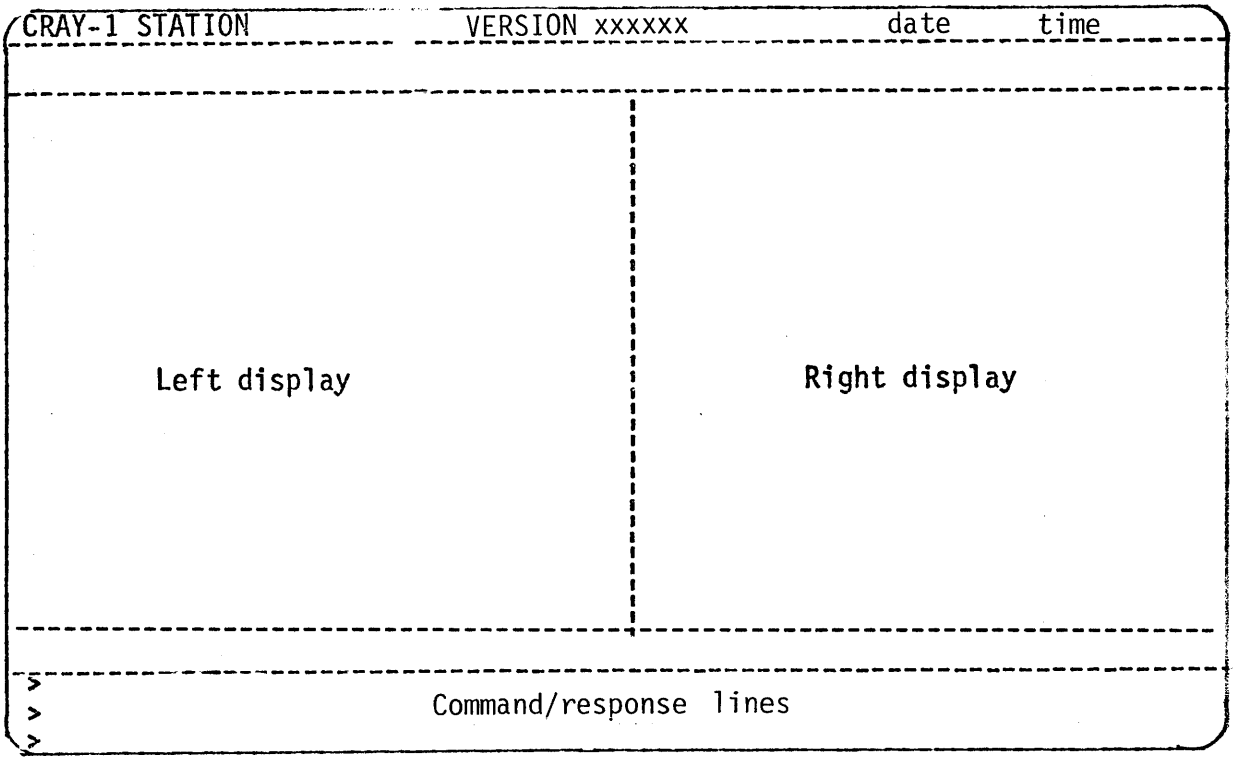


Figure 4.3-1. Debug display format

4.3.2 DEBUG DISPLAY COMMAND

The debug command that invokes displays for the right and left halves of the data screen consists of two characters, each of which represents a predefined display type.

Format:

$d_1 d_2 \cdot$ ↓

d_1 Specifies the contents of the left display.

d_2 Specifies the contents of the right display. If d_2 is omitted, the right display is clear.

4.3.3 DISPLAY ROLL COMMAND

The contents of a debug display can be advanced or regressed through a command having the following form:

$d \pm n$ ↓

d Identifies display to be rolled.

n Octal count of the number of lines the display is to be rolled forward (+), or rolled backward (-).

An attempt to roll an XP display is ignored.

As shown in table 4.3-1, special keys can also be used to roll displays.

Table 4.3-1. Special keys for controlling debug displays

Key	Function
+	Rolls left display forward 16 entries.
-	Rolls left display backward 16 entries.
>	Rolls right display forward 16 entries.
<	Rolls right display backward 16 entries.

4.3.4 DISPLAY - DISPLAY DEFINITION

FUNCTION: Describes the information to be displayed and the format in which it is to be displayed. It also assigns a display character to the display and invokes it.

FORMAT: DISPLAY,*d*,*info* {,*format*}{,*mode*} ↓

d Display character, A through Z, to be assigned to the display information.

info Identifies the type of information to be associated the display character. The parameter is required. The program with which the information is associated is determined by the *mode* parameter.

XP Exchange package. *format*, if specified, is ignored.

B*rr* B registers. The first register displayed is specified by *rr*, where *rr* is 0-77$_8$. The *format* parameter, if specified, is ignored.

T*rr* T registers. The first register displayed is specified by *rr*, where *rr* is 0 - 77$_8$.

V*ree* V register elements. Elements of V register *r* are displayed where *r* is in the range 0 - 7. The first element displayed is specified by *ee*, where *ee* is in the range 0-77$_8$. If only one or two digits are supplied, for *ree*, *r* is assumed to be 0 and the digits specify the first element number.

address Memory information. *address* is the address of the first word or parcel to be displayed. It can be a word address with a suffix (A, B, C, or D) to indicate the relative parcel within the word or can be a parcel pointer (a word address times four plus the offset of the parcel within the word, i.e., 0, 1, 2, or 3) with a P suffix. If a parcel address is specified and the format is not INSTRUCTION, the displays begin with the word containing the referenced parcel.

format Format for the displayed information. If this parameter is omitted, a default is used.

WORD Each line consists of a word address followed by 22 octal digits and the ASCII interpretation of the octal value.

WORD is the default format for *address* information.

PARCEL Each line consists of a word address and four octal groups of 16 bits each.

FLOAT Each line consists of a memory word address, one digit representing the sign of the coefficient, five octal digits representing the exponent, and four groups of four octal digits each representing the coefficient.

FLOAT is the default format for T*rr* and V*ree*.

INSTRUCTION

Each line consists of a memory address and four groups of octal digits representing 16- or 32-bit instructions. Parcels prior to the initial parcel address in the first word are blanked out. Single-parcel instructions are shown as 6 octal digits. Two-parcel instructions are shown as 12 octal digits.

XP Exchange package format.

This format is the default for XP information.

mode Program mode; the default is determined by the MODE command. Refer to table 4.3.2 for *mode* dependencie

EXECUTIVE Display data is associated with the system executive. All addresses are assumed to be absolute.

JOB {,*jsq*} Display data is associated with the job explicitly identified by *jsq* or by the MODE or the ASSIGN command if *jsq* is omitted from this command. *jsq* is the JSQ ordinal for the job. All addresses are assumed to be relative to the job's base address.

TASK {,*tn*} Display data is associated with the task explicitly identified by *tn*, or by the MODE or ASSIGN command if *tn* is omitted from this command. All addresses are assumed to be relative to the STP base address.

STATION Display data is associated with the statio

Table 4.3-2. Display mode dependencies

info	*mode*			
	EXECUTIVE	TASK	JOB	STATION
XP	NO	YES	YES	NO
B*rr*	NO	NO	YES	NO
T*rr*	NO	NO	YES	NO
V*ree*	NO	NO	YES	NO
address	YES	YES	YES	YES

TYPE: Debug

PREREQUISITES: None

EXAMPLES: Refer to section 4.3.6.

4.3.5 ALTER - ALTER DISPLAY DEFINITION

FUNCTION: Redefines selected parameters for a display described by a DISPLAY command.

FORMAT: ALTER ,*d* {,*info*}{,*format*}{,*mode*}

Refer to section 4.3.4 for descriptions of the *d* *info*, *format*, and *mode* parameters.

TYPE: Debug

PREREQUISITES: None

4.3.6 DEBUG - DEBUG DISPLAY DIRECTORY

FUNCTION: Activates the debug display directory. The directory shows the parameters associated with the display characters A through Z. Refer to section 4.3.7 for a display example.

FORMAT: DEBUG

TYPE: Debug

PREREQUISITES: None

4.3.7 EXAMPLES OF DISPLAYS

1. An exchange package display is assigned the designator A. Format is XP by default (the format parameter is not applicable) and mode is TASK 4.

```
DIS A X ,, T 4
P    46446D        A0 77777714
BA   20000         A1       100
LA  126000         A2        14
M       12         A3        14
XA     340         A4     26373
VL     100         A5     27055
F        0         A6         2
B0 0046517D        A7     17613

S0 000000 0000 0000 0000 0000
S1 000000 0000 0000 0000 0000
S2 000000 0000 0000 0000 0000
S3 000000 0000 0000 0000 0000
S4 000000 0000 0000 0000 0000
S5 000000 0000 0000 0005 1742        S
S6 000000 0000 0000 0000 0004
S7 000000 0000 0000 0000 0003

>DIS A X X T 4
>SNAP SNAP
```

2. Memory starting with word 200 is displayed in word format and EXEC mode. The display is assigned the designator A.

```
DIS A 200 W E

    200 0000000000000000000203
    201 0003777777600000000260
    202 1001770000000000000000
    203 0000060020000100000000
    204 1000100040000200000000
    205 1000140060000300000000
    206 1000020100000400000000
    207 1000040120000500000000
    210 1000050140000600000000
    211 1000120160000700000000
    212 0000000000000000000000
    213 0000000000000000000000
    214 0000000000000000000000
    215 0000000000000000000000
    216 0000000000000000000000
    217 0000000000000000000000

>DIS A 200 W E
>SNAP SNAP
```

3. Memory starting with word 27005 is displayed in word format and TASK mode. The display is assigned the designator A.

```
DIS A 27005 W T

  27005 042104132304711323206З DD-19-43
  27006 020000300000023000021563        #
  27007 000633024000220000266441        -!
  27010 000000000000000000000000
  27011 000000000000000000000000
  27012 000000000000000000000000
  27013 000000000000000000000000
  27014 000000000000000000000000
  27015 000000000000000000000000
  27016 000000000000000000000000
  27017 000000000000000000000000
  27020 000000000000000000000000
  27021 000000000000000000000000
  27022 000000000000000000000000
  27023 000000000000000000000000
  27024 000000000000000000000000

>DIS A 27005 W T
>SNAP SNAP
```

4. The following displays memory starting with word 200 in parcel format and EXEC mode.

```
DIS A 200 P E

    200 000000 000000 000000 000203
    201 000377 177774 000000 000260
    202 100177 000000 000000 000000
    203 030006 000400 000400 000000
    204 100010 001000 001000 000000
    205 100014 001400 001400 000000
    206 100002 002000 002000 000000
    207 100004 002400 002400 000000
    210 100005 003000 003000 000000
    211 100012 003400 003400 000000
    212 000000 000000 000000 000000
    213 000000 000000 000000 000000
    214 000000 000000 000000 000000
    215 000000 000000 000000 000000
    216 000000 000000 000000 000000
    217 000000 000000 000000 000000

>ALTER A,,P
>SNAP SNAP
```

5. This example is of memory starting with parcel address 60000A in instruction format and EXEC mode.

```
DIS A 60000A I E

0060000 1216000000077  043701 044167
0060001 055101 045012 015 00040021B
0060002 123200000007  055234 023520
0060003 020600005000  030616 12630000
0060004  0103  042450 044134 04050000
0060005  0200  060225 023720 023410
0060006 031047 013 176400068 12630000
0060007  0101  043430 055460 054230
0060010 050324 055250 136300000101
0060011 020700000200  030667 022710
0060012 030337 030556 022704 12310000
0060013  0000  123200000001  12330000
0060014  0002  123400000003  030337
0060015 030667 031065 136117777774
0060016 136217777775  136317777776
0060017 136417777777  013 00040012D

>DIS A 60000 I E
>SNAP SNAP
```

6. The following illustrates a left display of memory starting at word address 7000 in floating point format and a right display of the same area in word format. The mode is EXEC. The displays are designated as A and B.

```
DIS A 7000 F E

7000 004000 0000 0013 4704 0120
7001 000000 0000 0000 0010 4000
7002 000000 0000 0000 0000 0000
7003 000000 0000 0000 0000 0000
7004 001000 0000 0013 4702 0120
7005 000000 0000 0001 1216 6000
7006 000002 0004 3665 0006 7246
7007 020000 0000 0013 4670 0120
7010 007000 0000 0013 4702 0120
7011 000000 0000 0000 0005 5000
7012 041461 2303 0401 2001 3400
7013 000000 0000 0000 0000 1000
7014 016000 0000 0013 4702 0120
7015 000000 0000 0000 0005 3000
7016 000000 0000 0000 0000 0000
7017 000000 0000 0000 0000 0000
```

```
DIS B 7000 W E

7000 00400000000001347040120        @P
7001 00000000000000000104000
7002 00000000000000000000000
7003 00000000000000000000000
7004 00100000000001347020120         P
7005 00000000000000112166000       (
7006 00000200043665000067246     G
7007 02000000000001346700120         P
7010 00700000000001347020120         P
7011 00000000000000000055000        Z
7012 04146123030401200013400 CiL1 @
7013 00000000000000000001000
7014 01600000000001347020120       P
7015 00000000000000000053000        V
7016 00000000000000000000000
7017 00000000000000000000000
```

```
>AB.
>SNAP SNAP
```

7. Display C provides B register information. The format parameter is not applicable; the default format is shown below. B registers for the job with the jsq of 3 starting with register B20 are displayed.

```
DIS C B20 ,, J 3

    B20 00011101 0002220B
    B21 00011125 0002225B
    B22 00011525 0002325B
    B23 00000505 0000121B
    B24 00000000 0000000A
    B25 00000000 0000000A
    B26 00000000 0000000A
    B27 13000000 2600000A
    B30 00000000 0000000A
    B31 00000000 0000000A
    B32 00000000 0000000A
    B33 40002200 0000440A
    B34 03242111 0650422B
    B35 14030060 3006014A
    B36 24442506 5110521C
    B37 20627015 4145603B

>C.
>SNAP SNAP
```

8. The left display, S, shows the contents of elements 0 through 17 of V register V0 in floating point format. The information is for the job having the jsq of 7. The right display, T, shows the contents of T registers 0 through 17 in floating point format for the same job.

```
DIS S V0 F J 7
VM=0000006610120016005000 VL=003
V0 00 063000 0000 0133 3526 3077
   01 113000 3444 7145 5771 2300
   02 075304 3120 0000 0005 5205
   03 062077 3720 0224 2726 2000
   04 000000 2667 2143 1763 3400
   05 075236 3060 0000 0005 5204
   06 061000 0000 0133 3466 0477
   07 044640 5426 1541 0000 0000
   10 055203 3000 0000 0011 4421
   11 060277 2232 0172 4746 0177
   12 047000 3651 7100 0000 0000
   13 134773 1000 0000 0010 5673
   14 014000 0000 0134 1301 0000
   15 000000 3666 6010 0000 0000
   16 100644 0000 0000 0010 0262
   17 061777 0000 0312 1150 0030

>ST.
>SNAP SNAP
```

```
DIS T T0 F J 7

    T00 000000 6610 1200 1600 5000
    T01 154202 4000 4000 0004 3403
    T02 100110 0000 0267 1470 0210
    T03 000000 2624 3200 6200 0000
    T04 056024 4020 4000 0007 5304
    T05 100510 0000 0330 4110 0610
    T06 000000 2210 6601 6200 0000
    T07 153232 4040 4000 0013 4006
    T10 101110 0000 0267 7570 1210
    T11 000000 4341 5202 6205 5000
    T12 055415 4060 4000 0007 2746
    T13 101510 0000 0165 7310 1610
    T14 067000 2647 1203 6210 6000
    T15 055162 4100 4000 0004 7711
    T16 102070 4040 0140 7570 2110
    T17 000000 2375 2204 4200 0000
```

9. In the following example, elements 20 through 37 of V register 0 for the job with jsq of 3 are shown in word format.

```
DIS C V20 W J 3
VM=125135002047441600041Ø VL=003
V0 20 0776605374434762000003
   21 1244002260052300400402     K S
   22 0004036103406103000402      81
   23 0004036303406103200402      81
   24 0004056503405162102670      8)
   25 0776451023450300732070     !9C 8
   26 0012346400020400106610
   27 1066104030400100400403
   30 1220701420660227200407    81  ]
   31 0154030037244112041403       !(C
   32 1514040037040100400403
   33 1260701420640100400402    81
   34 1524001060130776010000     #
   35 1010050020374416001001        8
   36 1537700000632201000417
   37 1024005240240103102520           P

>
>SNAP SNAP
```

10. The following is an example of the DEBUG display directory. Parenthetical items indicate the defaults for the display mode and the job or task ordinal.

```
DEBUG DISPLAY DIRECTORY                 REFRESH OFF 60
                                        MODE     T
()= DEFAULT                             ASSIGN   T  0
* = ILLEGAL DISPLAY REQUEST             ASSIGN   J  0

DIS A 0 P S                             DIS N V000 P J (0)
DIS B B00 ,, J (0)                      DIS O V000 W J (0)
DIS C 0 W J (0)                         DIS P 0 P J (0)
DIS D 0 W T                             DIS Q 0 P T
DIS E 0 W E                             DIS R 0 P E
DIS F 0 F J (0)                         DIS S 0 P S
DIS G 0 F T                             DIS T T00 F J (0)
DIS H 0 F E                             DIS U 100 X E
DIS I 0A I J (0)                        DIS V V000 F J (0)
DIS J 0A I T                            DIS W 0 W J (0)
DIS K 0A I E                            DIS X X ,, J (0)
DIS L T00 P J (0)                       DIS Y X ,, T (0)
DIS M T00 W J (0)                       DIS Z 0 X E

>DEBUG
>SNAP
```

UNB UTILITY PROGRAM 5

5.1 INTRODUCTION

The UNB utility program converts the absolute binary load module contained on the next record of the input dataset to an integral multiple of 512 words and writes them on the output dataset (i.e., appends them to the dataset). The input dataset is CRAY-1 blocked format; the output dataset is unblocked.

UNB replaces the program descriptor table (PDT) and text table header in the input record with an 8-word prefix (figure 1-1). Thus, the resulting binary image appended to the output dataset is similar to an absolute binary load module generated by Data General CAL on the Eclipse.

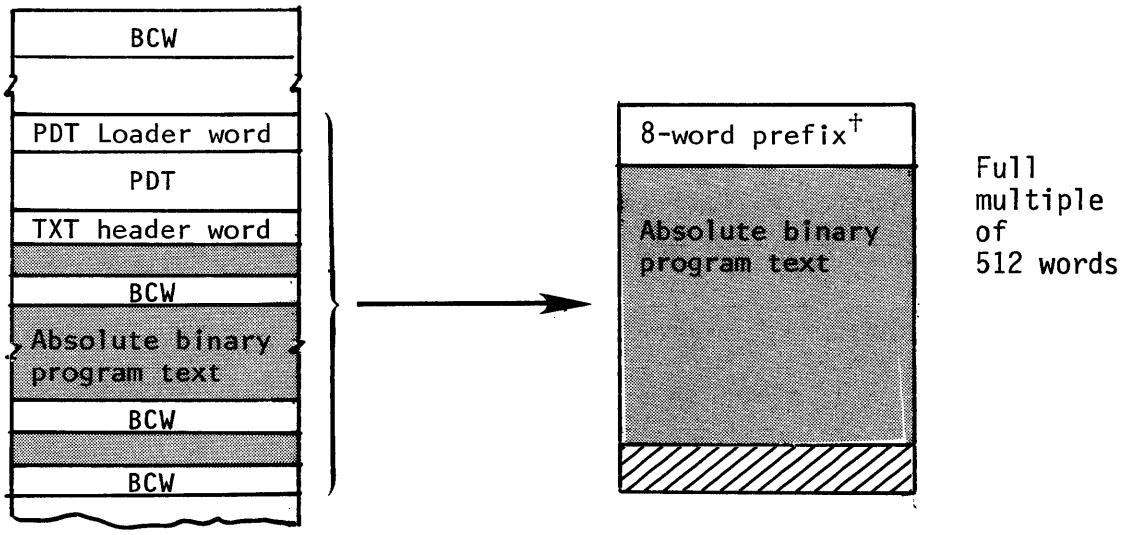


Figure 5-1. Unblocked binary format

5.2 UNB CONTROL STATEMENT

UNB is loaded into the user field and executed as a user job step when an UNB control statement is encountered. Parameters on the control statement identify the input and output datasets.

Format:

```
UNB (I=idn,O=odn)
```

† Each call to UNB produces a prefix; thus a prefix precedes each absolute binary program text.

Parameters:

I=*idn* Name of input dataset. The input dataset, which is a CRAY-1 blocked dataset, must be positioned prior to a binary record containing the next absolute load module to be converted. The default for *idn* is $BLD.

O=*odn* Name of output dataset. The default for *odn* is OUT. There must not be an ASSIGN control statement for this dataset. If dataset *odn* is not empty, UNB positions it after the last block

5.3 RESTRICTIONS

Sufficient memory must be available to a job using UNB so that the user field is large enough to contain a buffer for the input record. For example, in generating an operating system binary, approximately $100{,}000_8$ words must be available for this buffer.

If sufficient memory is not available to unblock the record, the following error message is issued:

NOT ENOUGH MEMORY TO CONTAIN THE RECORD.

UNB checks the fatal error flag in the PDT before replacing the PDT with the 8-word prefix. If the flag is set, UNB aborts the job step. This allows processing of multiple source files without aborting until the latest possible step. By removing the ABORT keyword from the CAL statement, for example, EXEC, STP, and CSP may all be assembled and assembly errors (if any) listed for each. The combined binary will be generated by UNB only if all three are error-free. However, if an unblocked binary is desired despite assembly errors, the DEBUG keyword should be added to the CAL statement (see CAL Reference Manual, CRI pub. no. SR-0000).

CRAY-1 SIMULATOR (CSIM) 6

6.1 INTRODUCTION

The CRAY-1 Simulator (CSIM) resides on CRAY-1 mass storage as a permanent dataset and is loaded and executed as a result of a CSIM control statement being encountered in a job's control statement file.

The simulator is a powerful diagnostic tool that allows several system analysts each to concurrently exercise and monitor the performance of their own development versions of the CRAY-1 Operating System. This activity occurs in a batch environment without affecting the running, production-level operating system. Figure 6-1 illustrates the relationship of the simulated operating system and the running version of the operating system.

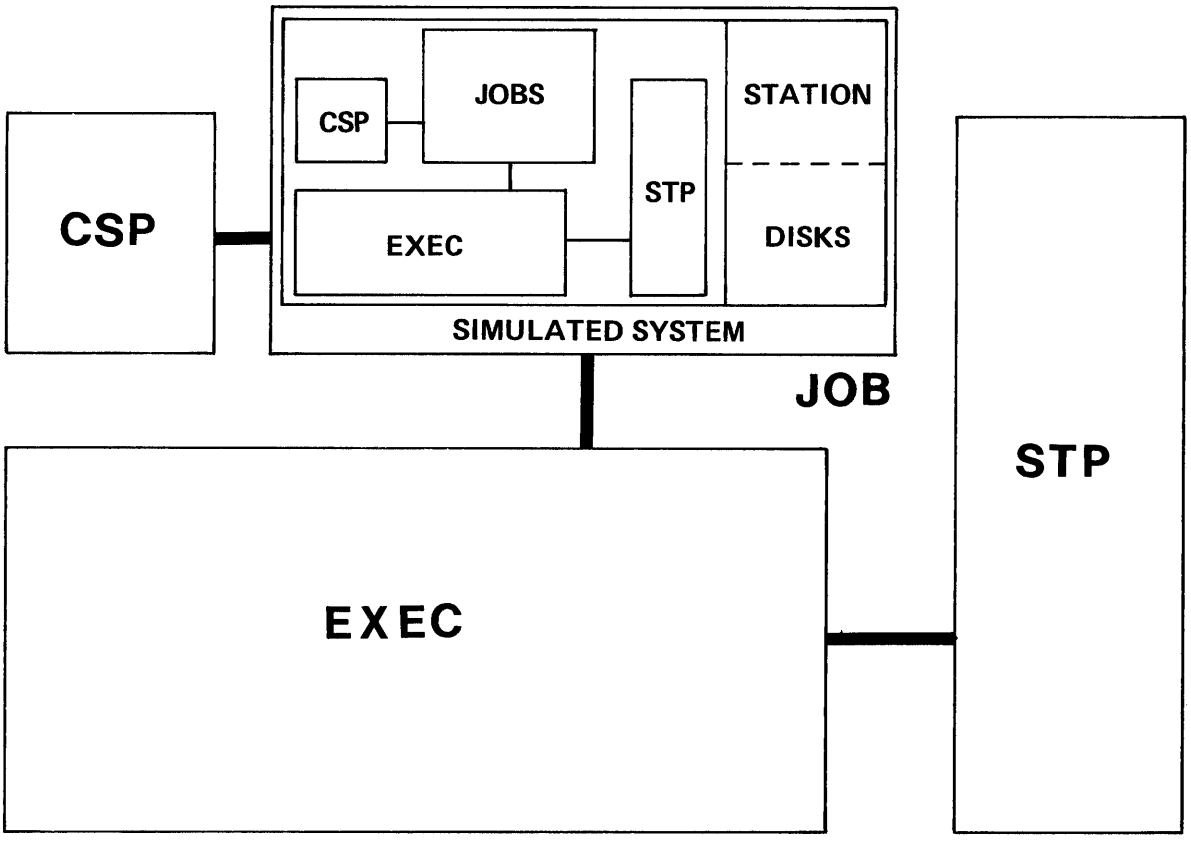


Figure 6-1. Relationship of CSIM to CRAY-OS

The simulator may be regarded as a laboratory test instrument that serves a system analyst much as an oscilloscope serves an electrical engineer. The analyst controls input to the simulated system, tracks performance, and observes reactions at selected points. Thus, simulation is an analyst's way of putting the operating system "on the bench".

To understand and effectively use the simulator, the user should be familiar with the external features of the CRAY-OS Operating System as described in CRI publication 2240011 and with its internal organization as described in Part 1 of this manual.

6.1.1 SIMULATED CONFIGURATION

The following processors and storage areas are simulated:

- a CRAY-1 CPU,
- a DCU-2 controller connected to the CPU on channels 4 and 5,
- a front-end processor connected to the CPU on channels 2 and 3,
- one million words of bipolar memory, and
- two DD-19 disks connected to the DCU-2 controller.

Figure 6-3 is a diagram of the configuration

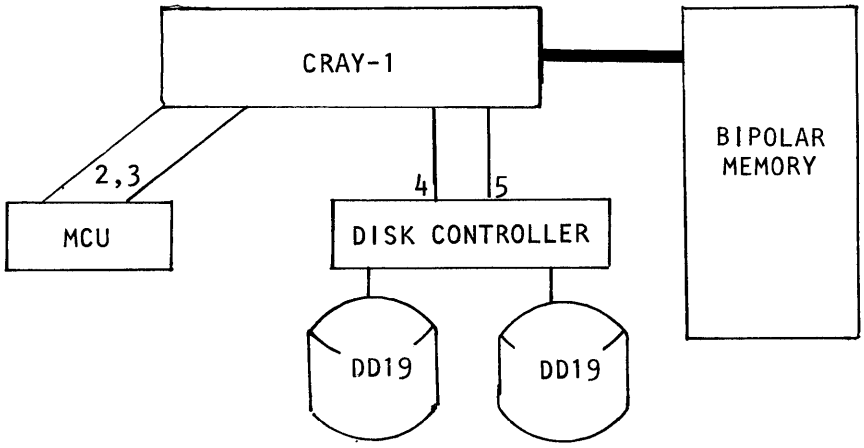


Figure 6-3. Simulated CRAY-1 configuration

6.1.2 CRAY-1 CPU SIMULATOR

The CPU simulates instruction execution for all standard instructions. The process differs somewhat from that of an actual CRAY-1 in that instruction executions do not overlap. Each instruction completes before the next is interpreted. All operating registers visible to the user are simulated, as are the CA and CL register sets.

6.1.3 DCU-2 Controller

This simulation permits the following controller functions:

- Unit reserve
- Unit release
- Read
- Write
- Cylinder select
- Subsystem status

The simulator provides read/write response and subsystem status. The subsystem status is always returned as clear.

All processor operations are performed synchronously with instruction interpretation; that is, no CPU instruction is interpreted until the operation on the previous instruction is complete.

6.1.4 FRONT-END PROCESSOR

The simulated front-end processor can perform the following operations:

- Log on
- Send user dataset
- Send input dataset
- Receive output dataset

All data transfers across the channel are accomplished synchronously with CPU instruction interpretation.

6.1.5 BIPOLAR MEMORY

Bipolar memory is disk-resident. It is managed by a virtual memory algorithm to provide better response time than a disk-based memory while keeping the simulator small. To change the size of resident memory, CSIM must be regenerated with a revised value for the parameter CORE1.

6.1.6 DISK STORAGE

The disk simulation routines read or write simulated local datasets on the actual CRAY-1 mass storage. The simulated configuration allocates space on two simulated disks. (To CSIM, each of these "disks" is a dataset; one is named DD190 and the other is named DD191. A parameter of the CSIM control statement sets the physical limits for these datasets to provide a limit to the disk space that CSIM can acquire for a dataset during simulation.) The simulated disks connect to a simulated controller on simulated channels 4 and 5. (see figure 6-3.)

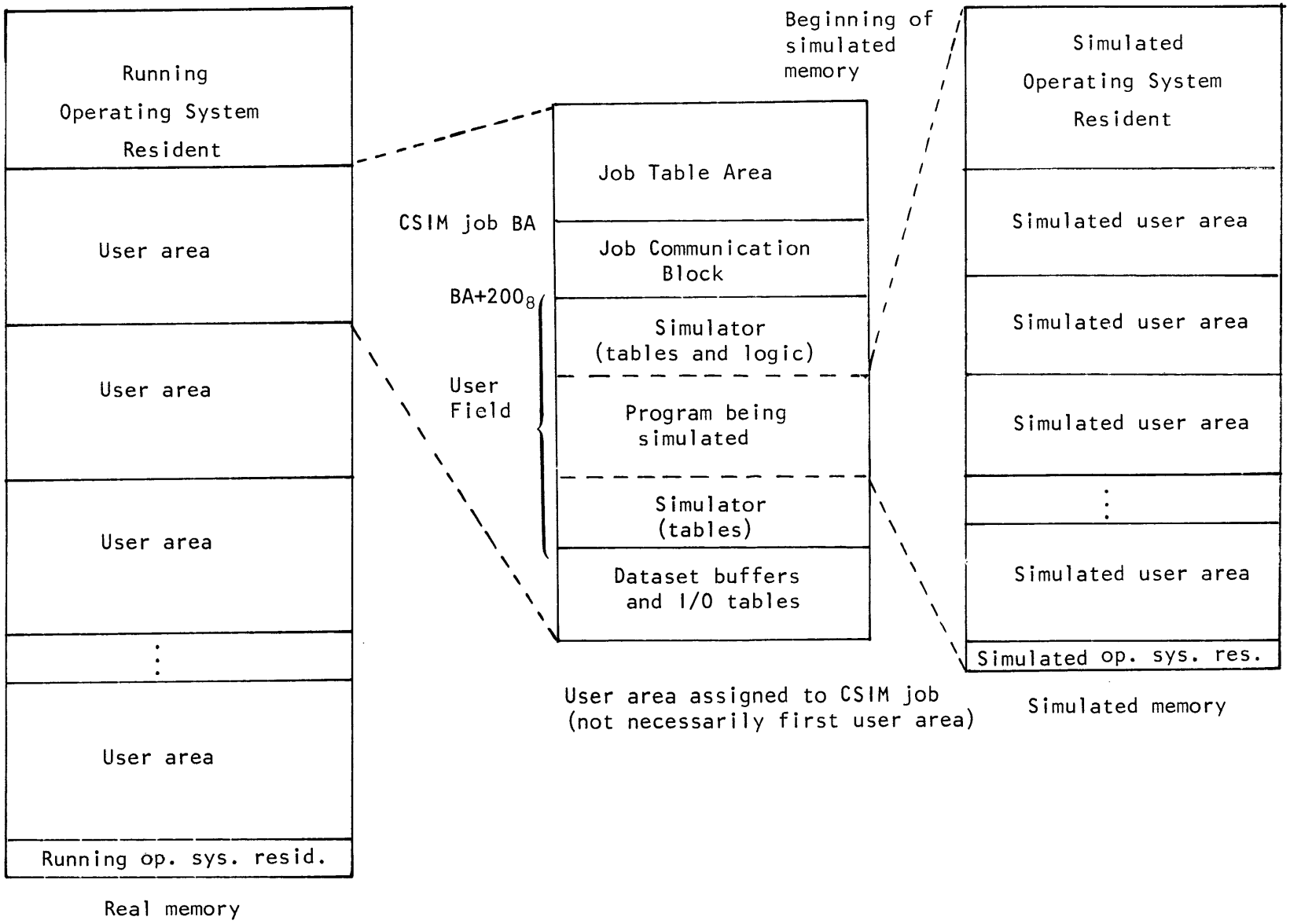


Figure 6-2. Memory assignment of CRAY-1

6.2 CSIM EXECUTION

The simulator consists of a control program, an instruction interpreter, and a set of interface simulation routines. This software is loaded as a result of the CSIM control statement being encountered in the user job deck. The simulator begins execution by reading a set of simulator directives telling it such information as what version of the operating system to load or restart, conditions to be monitored, and when performance data is to be written on the list dataset. These directives are described later.

6.2.1 CRAY-OS SIMULATION

The version of the operating system to be simulated is loaded immediately after the simulator software (figure 6-2), by that software. Some work area for the simulator follows the operating system. The interpreter begins "executing" instructions at a point determined by the exchange package at address 0 in EXEC, just as if deadstarting COS on the real CRAY-1. This simulated "execution" consists of the CSIM interpreter simulating the results of instruction execution instruction by instruction. If the instruction to be "executed" is an exchange sequence, for example, contents of simulated registers are changed accordingly and the next instruction to be "executed" is determined by the exchange sequence. If CSIM directives have previously noted such a change of register contents as a dump condition, then the interpreter calls the routines that perform dumps before "executing" the next instruction. If an instruction initiates an I/O operation, the interpreter calls routines that simulate the devices on that channel.

6.2.2 CHECKPOINTS OF THE SIMULATED SYSTEM§

A simulation can be stopped at any point in execution via the STOP directive and checkpointed via the CPT directive.

This feature enables long simulations to be broken up into several shorter runs. Also, work on Startup logic of COS is made more convenient, e.g., one can use a checkpoint after an Install to check out deadstart logic. Additionally, some time can be saved by using a checkpoint taken after Startup of COS to avoid redoing the work to Startup.

6.2.3 INPUT OF JOBS AND DATA

Jobs and other datasets (for example, the dataset containing the operating system and the dataset containing the parameter file) are staged to CRAY-1 mass storage as staged (ST) permanent datasets and are associated with the CSIM job through ACCESS control statements. Station commands (processed by SKOL) identify which of these datasets are jobs and which are data for jobs.

The $OUT dataset for a simulated job is staged out with the name of the simulated job as taken from its JOB statement.

6.3 CSIM CONTROL STATEMENT

The control statement that calls the simulator has the following format:

```
CSIM,L=ldn,I=idn,LINES=n,T=tl,OPSYS=osdn,OSPAR=pfdn,MAXBK=ms,VMEM=vm.
```

§ Deferred implementaion

L=*ldn* Specifies the dataset that is to receive the list output (such as dumps, messages, etc.). The default is $OUT.

I=*idn* Specifies the input dataset, which contains CSIM directives. The default is $IN.

LINES=*n* Specifies the maximum number of lines to be printed. The default for *n* is 1000. *n* is assumed to be a decimal value.

T=*tl* Specifies the time limit for the simulator program. The value is specified in decimal. The default is 10 seconds of CPU time. When the simulator determines there is no more work to be done, it stops.

OPSYS=*osdn* Specifies the dataset containing the version of the operating system to be deadstarted. Omission of the parameter gives a default of COS; simply specifying OPSYS is illegal.

OSPAR=*pfdn* Defines startup parameter file. Specifying only OSPAR gives a default of COSPAR. No parameter file is required when this parameter is omitted. The system acts as if there were a parameter file containing the two parameters *INSTALL and *END.

If the form OSPAR=*pfdn* is used, then *pfdn* contains the parameter file.

MAXBK=*ms* This parameter limits the number of sectors that will be used by CSIM for each of its simulated disks. That is, in terms of CSIM, it limits the sizes of datasets DD190 and DD191 on which CSIM maintains all of the simulated datasets. The default is the capacity of an entire DD-19 in the actual CRAY-1 system. *ms* is specified in decimal.

VMEM=*vm* This parameter enables or disables the simulation of virtual memory. When *vm* is 0, there is no virtual memory management, and simulator memory size is $200{,}000_{10}$. When VMEM is omitted or *vm* is non-zero, simulated memory size is 1 million words, although only $100{,}000_{10}$ actual words may be present in job memory at any one time.

Example:

```
CSIM,LINES=5000,T=100,OPSYS=TESTCOS,OSPAR=TESTPAR,MAXBK=150,VMEM=2.
```

6.3.1 SIMABORT CONTROL STATEMENT

The SIMABORT control statement unconditionally stops CSIM execution. CSP execution of SIMABORT on the actual CRAY-1 hardware is treated as a no-op.

Format:

```
SIMABORT.
```

6.4 GENERAL FORM AND SYNTAX OF CSIM DIRECTIVES

The CSIM directives are a file on a dataset identified by the I parameter on the CSIM control statement.

A directives file contains one or more sets of directives for defining conditions under which the simulated operating system execution is to be observed. The START directive, which initiates simulated execution, marks the end of each directive set.

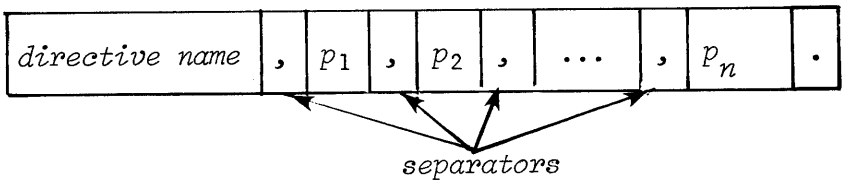
The user can monitor STP, EXEC, or user job execution and can request periodic or one-time dumps of information based on conditions such as:

- Execution of a specific instruction or range of instructions,
- A register value change,
- A value being reached,†
- An error condition,
- A memory location being read or written, and so on.

At the completion of one set of directives, a user can restart execution with a different set of criteria.

6.4.1 GENERAL FORM OF DIRECTIVES

A CSIM directive has the following general form:



directive name The directive name begins in column 1 and is terminated by a separator.

separators Parameters are separated from each other and from the directive name by a comma. Embedded blanks are not permitted within a parameter.

p_i Parameters are defined for each directive. A numeric value is assumed to be octal unless explicitly specified otherwise using one of the following forms:

D'*value*†	Decimal
O'*value*†	Octal (default)
'*value*'L†	ASCII, left adjusted with zero fill
'*value*'R†	ASCII, right adjusted with zero fill
'*value*'H†	ASCII, left adjusted with blank fill

† Deferred implementation

Parameters are in keyword form and can be in any order. A parameter is required unless otherwise indicated in its description.

A period terminates the parameter portion of a directive. Comments can follow the period to the 80th character. An end-of-file or an end-of-data terminates the directive file.

6.4.2 DIRECTIVE PARAMETERS

Each directive description mentions all parameters for the directive. This section illuminates several complex parameters in detail so that only a short comment need be made in the directive description itself.

N1,N2,I versus E1,E2,I

Some CSIM directives have these types of iteration control. The user either selects N1,N2,I parameters or E1,E2,I parameters; N and E cannot be mixed. E (as in entry) refers to how many times a P1,P2 range is entered, regardless of how much printout (e.g., how many TRACEs) occurs on each entry. N refers exactly to instances of printout.

Example:

(1) TRACEJP,M=U,P1=5A,P2=100C,E1=5,E2=11,I=2.

This directive will print for each jump instruction found between P=5A and P=100C in user mode on the 5th, 7th, 9th, and 11th entries into that P-range. Since no jump instruction might be found, no printout might occur for these four entries into the P-range, but a loop could cause thousands of lines for only one of these entries.

(2) TRACEJP,M=U,P1=5A,P2=100C,N1=5,N2=7,I=1.

This directive ensures that even if there are thousands of jump instructions in the P-range specified, printout will occur only for the 5th, 6th, and 7th jump instructions ever encountered in that range. Thus, just these printouts will occur, despite many entries into that P-range.

CAUTION

Whenever P1 is used, P2 must also be used. The same is true for N1 and N2 and for E1 and E2.

ON,OFF,TAG

The TAG parameter assigns a numeric label or name to its directive. Thus, the directive can be turned on or off respectively by use of an ON or OFF parameter equated to the TAG value. A directive containing a TAG parameter is initially OFF.

The directives in the following example act together to cause jump instructions of a portion of EXEC (16400C through 17001D) to be traced after the second time a certain range of user memory addresses (15 through 17) is written by the user P-range 0A through 177777D.

```
TRAP,M=U,P1=0A,P2=177777D,RF=W,A1=15,A2=17,N1=2,N2=2,I=1,ON=5.
TRACEJP,M=E,P1=16400C,P2=17001D,TAG=5.
```

A directive can turn itself off.

The following directives act together to trace only the instructions by which the user program is entered and by which STP is entered (in each case from EXEC).

```
TRACE,M=E,P1=0A,P2=3777777D,ON=1,OFF=2.
TRACE,M=U,P1=0A,P2=3777777D,ON=2,OFF=1,TAG-1.
TRACE,M=S,P1=0A,P2=3777777D,ON=1,OFF=2,TAG=2.
```

The values for ON, OFF, and TAG must be between 0 and $2^{11}-2$.

6.4.3 MACROS

To provide CSIM users location-keyed directives, CSIM supports a set of macros that may be assembled into a user's code. On the actual CRAY-1, the 0017jk instruction is a no-op. However, for CSIM it is an escape mechanism enabling a program to signal CSIM so that directives are enabled or disabled and messages are output. The following macros generate the 0017jk instruction in various forms. These macros are in both COSTXT and $SYSTXT.

- Enter message in CSIM output dataset

Location	Result	Operand
	SIMMSG	R1

where R1 is an A or S register that contains the starting address of a message terminated by a null byte.

- Abort simulator

Location	Result	Operand
	SIMABORT	

- Enter RTC in CSIM output

Location	Result	Operand
	SIMRTC	

- Disable CSIM directive execution

Location	Result	Operand
	DISDIR	

- Re-enable execution for all CSIM directives that are turned ON

Location	Result	Operand
	ENDIR	

- Enable directive execution for directives with TAG=N (turning them ON)

Location	Result	Operand
	TAGON	N

- Disable directive execution for directives with TAG=N (turning them OFF)

Location	Result	Operand
	TAGOFF	N

6.5 CSIM DIRECTIVES

6.5.1 DS - SIMULATED DEADSTART

The DS directive causes the simulator to load the operating system from the dataset identified by the OPSYS parameter on the CSIM control statement. The system is loaded into simulated memory location 0 (absolute) where it is ready to begin simulated execution.

Format:

```
DS.
```

6.5.2 CPT - CHECKPOINT§

This directive takes a checkpoint of simulated memory, register, disk I/O, channel registers, and station. The checkpoint information is written on a dataset defined by the directive.

Format:

```
CPT,DN=dn,L=lwa.
```

DN=*dn* Dataset on which checkpoint is written; default is CSIMCPT.

L=*lwa* Last word of simulated memory to be dumped; default is all.

NOTE

Instruction interpretation must be stopped before a checkpoint is taken.

6.5.3 RST - RESTART§

This directive causes the simulator to load the checkpointed system from the dataset specified by a parameter of the directive. It does not resume simulated execution. Simulated memory, registers, disk, and the station are recovered.

---

§ Deferred implementation

Format:

```
RST,DN=dn.
```

DN=*dn* Defines dataset containing checkpointed system. Default is CSIMCPT.

6.5.4 EXIT - SIMULATOR ERROR EXIT

If an error such as CSIM time limit exceeded occurs during simulation, the simulator terminates the simulation and looks in the set of directives for an EXIT directive. Following the EXIT directives may be directives requesting static dumps of the simulated program area. Simulation cannot be restarted when a fatal simulation error occurs. If there is no EXIT directive, the simulator terminates when it reaches the end of the directives.

Format:

```
EXIT.
```

6.5.5 STOP - STOP INSTRUCTION INTERPRETATION

This directive defines conditions under which instruction interpretation is suspended at a given location, perhaps so that a checkpoint can be taken or a new set of simulation conditions can be defined (i.e., the simulator will read a new set of directives). Simulated instruction interpretation can be resumed with START.

Format:

```
STOP,M=mode,P1,p1,P2=p2, {N1=n1,N2=n2,} I=i,ON=on,OFF=off,TAG=t.
                         {E1=e1,E2=e2,}
```

M=*mode* Mode of P address:

- E EXEC (monitor)
- S STP
- U User (default)
- X Stop regardless of state (relative to current base address)

P1=p_1 P2=p_2 Parcel location of instruction at which to stop (relative to mode).

N1=n_1 E1=e_1 See section 6.4.2 This optional parameter defines the first time to stop on this condition. A number of up to 48 bits can be specified. The default is 1 if no value is given for the keyword.

N2=n_2 E2=e_2 See section 6.4.2. This optional parameter defines the last time to stop on this condition. A number of up to 48 bits can be specified. The default is 2^{48}-1 if no value is given for the keyword.

I=i Increment. Stop is to be made every ith time this condition is encountered. A number of up to 48 bits can be specified. This parameter is optional; its default is 1.

ON=on Turns on each directive having TAG=on. See section 6.4.2.

OFF=off Turns off each directive having TAG=on. See section 6.4.2.

TAG=t Assigns a numeric label to provide control by an ON or OFF parameter. See section 6.4.2.

6.5.6 START - BEGIN INSTRUCTION INTERPRETATION

This directive marks the end of a directive set and initiates instruction interpretation after a DS or RST directive or resumes instruction interpretation subsequent to a STOP directive condition being met. Simulated execution begins at the address specified by the contents of the simulated P register.

Format:

```
START.
```

6.5.7 MEM - ENTER VALUE INTO SIMULATED MEMORY

This instruction initializes one or more simulates memory locations on the specified values. Normally, a MEM directive is issued only after a DS directive.

Format:

MEM,M=*mode*,P1=p_1,P2=p_2,A1=*fwa*,A2=*lwa*,S@=*bit*,N@=*bits*,VAL=*value*,
$\left\{\begin{matrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{matrix}\right\}$,I=*i*,C=*cdx*$_1$*cj*$_1$*cdx*$_2$*cj*$_2$...*cdx*$_n$,ON=*on*,OFF=*off*,TAG=*t*.

M=*mode* — Address mode:
- E EXEC (monitor)
- S STP
- U User (default)

P1=p_1 / P2=p_2 — Parcel address of instruction at which memory initializing is to occur; a word address suffixed by A, B, C, or D. If P is present, the memory initializing is conditional, and I and C have meaning.

A1=*fwa* — *fwa* defines the first or only memory location to be changed. This parameter is required. Addresses are absolute.

A2=*lwa* — *lwa* defines the last location to be changed. If this parameter is omitted, only the location defined by A1 is changed. Addresses are absolute.

S@=*bit* — *bit* is the number of the first bit of the field to be initialized within each memory location. The default is bit 0.

N@=*bits* — *bits* is the length of bits of the field to be initialized within each memory location. The default is 64.

VAL=*value* — *value* is set into each memory location (specified by A1,A2) within the designated field (specified by S@,N@), right adjusted.

N1=n_1 — See section 6.4.2. This optional parameter defines the

E1=e_1 first time to enter a value in simulated memory on this condition. A number of up to 48 bits can be specified. The default is 1 if no value is given for the keyword.

N2=n_2
E2=e_2 See section 6.4.2. This optional parameter defines the last time to enter a value in simulated memory on this condition. A number of up to 48 bits can be specified. The default is 2^{48}-1 if no value is given for the keyword.

I=i Increment. A value is to be entered into simulated memory every ith time this condition is encountered. A number of up to 48 bits can be specified. This parameter is optional; its default is 1.

C=$cdx_1cj_1cdx_2cj_2\ldots cdx_n$ §

This optional parameter defines conditions for initializing memory. Two or more conditions may be connected by a conjunction. Conditions and algorithms are evaluated from left to right.

cdx_i A condition has the following form:

$$\left\{\begin{matrix} reg \\ \\ (locn) \end{matrix}\right\} \quad op \quad \left\{\begin{matrix} reg \\ (locn) \\ value \end{matrix}\right\}$$

reg Register designator: An, Bn, Sn, Tn, or Vnee. For An, Sn, or Vnee, n is in the range 0 through 7. For Bn and Tn, n is in the range 0 through 77, ee is a 2-digit element number in the range 0 to 77. Equating a register to a parcel address value is not possible.

(locn) Designates contents of a given location

value An octal, decimal, or ASCII value. A floating-point value is assumed if D'*value* contains a decimal point.

---

§ Deferred implementation

op One of the following operators:

op	Significance
>	Greater than
<	Less than
=	Equal to
[	Less than or equal to
]	Greater than or equal to
#	Not equal to

cj_i Conjunction joining two conditions; one of the following:

conj	Operation (where A and B are conditions)
&	Logical AND of A and B
!	Logical OR of A and B
\	Exclusive OR of A and B
%	A AND NOT B
?	A OR NOT B
/	(A AND NOT B) OR (B AND NOT A)

ON=*on* Turns on each directive having TAG=*on*. See section 6.4.2.

OFF=*off* Turns off each directive having TAG=*on*. See section 6.4.2.

TAG=*t* Assigns a numeric label to provide control by an ON or OFF parameter. See section 6.4.2.

6.5.8 TRACE, TRACEJP, TRACEXP - DEFINE TRACE CONDITIONS

The TRACE, TRACEJP, and TRACEXP directives define trace conditions for a range of addresses.

The TRACE directive provides updates register information triggered by register value changes. Information reported consists only of current values for the register type receiving the change. Other information given includes the P register contents, the mode register contents, and the parcels comprising the instructions (the second parcel is 0 on a one-parcel instruction).

The TRACEJP directive defines the conditions under which all branch instructions within a given range are traced. For each branch instruction in the specified range, TRACEJP reports the parcel address of the branch instruction, the destination or fall-through parcel address, the mode, the contents of registers B0, A0, and S0, and the instruction itself.

The TRACEXP directive reports trace information consisting of the contents of the old and the new exchange packages, the contents of the P and mode registers, and the reasons for the exchange sequence each time an exchange sequence occurs within the designated range.

Trace information can be reported every time a condition is true or every ith time where i is an increment. The information can also be requested between the nth and mth times a condition is true, where n is the first occurrence to be reported and m is the last occurrence to be reported.

Output from the trace goes to the list output dataset for the simulator.

Formats:

TRACE,M=*mode*,P1=p_1,P2=p_2,$\left\{\begin{matrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{matrix}\right\}$,I=$i$,V,C=$cdx_1cj_1cdx_2cj_2\ldots cdx_n$,
ON=*on*,OFF=*off*,TAG=t.

TRACEJP,M=*mode*,P1=p_1,P2=p_2,$\left\{\begin{matrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{matrix}\right\}$,I=$i$,C=$cdx_1cj_1cdx_2cj_2\ldots cdx_n$,
ON=*on*,OFF=*off*,TAG=t.

TRACEXP,M=*mode*,P1=p_1,P2=p_2,$\left\{\begin{matrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{matrix}\right\}$,I=$i$,ON=*on*,OFF=*off*,TAG=$t$.

M=*mode* CPU mode:

- E EXEC (Monitor)
- S STP
- U User (default)
- Z Trace regardless of state (relative to current base address)

P1=p_1 Lower bound of range of addresses to be traced. This is a parcel address consisting of a word address and an A, B, C, or D suffix.

P2=p_2 Upper bound of range of addresses to be traced. This is a parcel address consisting of a word address and an A, B, C, or D suffix.

N1=n_1
E1=e_1 See section 6.4.2. This optional parameter defines the first time to trace on this condition. A number of up to 48 bits can be specified. The default is 1 if no value is given for the keyword.

N2=n_2
E2=e_2 See section 6.4.2. This optional parameter defines the last time to trace on this condition. A number of up to 48 bits can be specified. The default is 2^{48}-1 if no value is given for the keyword.

I=i Increment. Trace is to be taken every i^{th} time this condition is encountered. A number of up to 48 bits can be specified. This parameter is optional; its default is 1.

V§ Report vector register changes.

C=$cdx_1cj_1cdx_2cj_2...cdx_n$§
This optional parameter defines condition(s) for tracing. Two or more conditions may be connected by a conjunction. Conditions and conjunctions are evaluated from left to right.

---

§ Deferred implementation

cdx$_i$ A condition has the following form:

$$\left\{\begin{array}{l} reg \\ (locn) \end{array}\right\} \; op \; \left\{\begin{array}{l} reg \\ (locn) \\ value \end{array}\right\}$$

reg Register designator: An, Bn, Sn, Tn, or Vnee. For An, Sn, or Vnee, n is in the range 0 through 7. For Bn and Tn, n is in the range 0 through 77, ee is a 2-digit element number in the range 00 to 77. Equating a register to a parcel address value is not possible.

(locn) Designates contents of a given location.

value An octal, decimal, or ASCII value. A floating point value is assumed if D'*value* contains a decimal point.

op One of the following operators:

op	Significance
>	Greater than
<	Less than
=	Equal to
[	Less than or equal to
]	Greater than or equal to
#	Not equal to

cj$_i$ Conjunction joining two conditions; one of the following:

conj	Operation (where A and B are conditions)
&	Logical AND of A and B
!	Logical OR of A and B
\	Exclusive OR of A and B
%	A AND NOT B
?	A OR NOT B
/	(A AND NOT B) OR (B AND NOT A)

ON=*on* Turns on each directive having TAG=*on*. See section 6.4.2.

OFF=*off* Turns off each directive having TAG=*on*. See section 6.4.2.

TAG=*t* Assigns a numeric label to provide control by an ON or OFF parameter. See section 6.4.2.

Example:

```
TRACE,M=E,P1=200,P2=250,N1=1,N2=10,I=1,V,C=S1 S0&(1345)#0'77.
```

6.5.9 SNAPM, SNAPR, SNAPXP - DEFINE SNAP CONDITIONS

The SNAPM, SNAPR, and SNAPXP directives report memory, register, or exchange package contents based on execution of a specific instruction and, optionally, on a condition defined by the directive.

SNAPM directs the simulator to snap a limited range memory dump whenever a specific instruction is executed.

SNAPR directs the simulator to snap register contents whenever a specific instruction is executed.

SNAPXP directs the simulator to snap exchange package (and optionally register) contents when the specified exchange instruction is executed.

Output from snap goes to the list output dataset for the simulator.

Formats:

SNAPM,M=*mode*,P1=p_1,P2=p_2,$\begin{Bmatrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{Bmatrix}$,I=*i*,A1=*fwa*,A2=*lwa*,
C=*cdx*$_1$*cj*$_1$*cdx*$_2$*cj*$_2$...*cdx*$_n$,ON=*on*,OFF=*off*,TAG=*t*.

SNAPR,M=*mode*,P1=p_1,P2=p_2,$\begin{Bmatrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{Bmatrix}$,I=*i*,R=$r_1$:$r_2$:...:$r_n$,
C=*cdx*$_1$*cj*$_1$*cdx*$_2$*cj*$_2$...*cdx*$_n$,ON=*on*,OFF=*off*,TAG=*t*.

SNAPXP,M=*mode*,P1=p_1,P2=p_2,$\begin{Bmatrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{Bmatrix}$,I=*i*,R=$r_1$:$r_2$:...:$r_n$
C=*cdx*$_1$*cj*$_1$*cdx*$_2$*cj*$_2$...*cdx*$_n$,ON=*on*,OFF=*off*,TAG=*t*.

M=mode — CPU mode:

- E EXEC (monitor)
- S STP
- U User (default)
- Z Trace regardless of state (relative to current bass address)

P1=p_1
P2=p_2 — Parcel address of instruction at which snap is to occur; a word address suffixed by A, B, C, or D. If P is present, the snap is conditional and I and C have meaning.

N1=n_1
E1=e_1 — See section 6.4.2. First time to snap on this instruction. The default is 1 if no value is given for the keyword.

N2=n_2
E2=e_2 — See section 6.4.2. Last time to snap on this instruction. The default is 2^{48}-1 if no value is given for the keyword.

I=i — Increment. The snap is to occur every i^{th} time this instruction is executed. This parameter is optional; its default is 1.

A1=*fwa* — First address to be listed in output.

A2=*lwa* — Last address to be listed in output.

R=r_1:r_2:...:r_n† — Defines types of registers to be dumped. For SNAPR, can be A, B, S, T, V, VL, or VM; for SNAPXP, it can be B, T, V, VL, or VM.

C=$cdx_1cj_1cdx_2cj_2...cdx_n$† — This optional parameter defines condition(s) for snap. Two or more conditions may be connected by a conjunction. Conditions and conjunctions are evaluated from left to right.

cdx_i — A condition has the following form:

$$\left\{\begin{array}{l} reg \\ (locn) \end{array}\right\} \; op \; \left\{\begin{array}{l} reg \\ (locn) \\ value \end{array}\right\}$$

† Deferred implementation

reg Register designator: An, Bn, Sn, Tn, or Vnee. For An, Sn, or Vnee, n is in the range 0 through 7. For Bn and Tn, n is in the range 0 through 77. ee is a 2-digit element number in the range 00 through 77.

(locn) Designates the contents of a given location.

value An octal, decimal, or ASCII value. A floating point value is assumed if D'*value* contains a decimal point.

op One of the following operators:

op	Significance
>	Greater than
<	Less than
=	Equal to
[	Less than or equal to
]	Greater than or equal to
#	Not equal to

cj_i A conjunction joins two conditions and is one of the following:

conj	Operation (where A and B are conditions)
&	Logical AND of A and B
!	Logical OR of A and B
\	Exclusive OR of A and B
%	A AND NOT B
?	A OR NOT B
/	(A AND NOT B) OR (B AND NOT A)

ON=*on* Turns on each directive having TAG=*on*. See section 6.4.2.

OFF=*off* Turns off each directive having TAG=*on*. See section 6.4.2.

TAG=*t* Assigns a numeric label to provide control by an ON or OFF parameter. See section 6.4.2.

6.5.10 TRAP - DEFINE TRAP CONDITIONS

The TRAP directive causes the simulator to take a dump and optionally to terminate the simulation if a particular location or range of locations is referenced. The type of reference can be specified as read, write, or both.

Trap information can be reported every time a condition is true or every i^{th} time where i is an increment. The information can also be requested only between the n^{th} and m^{th} times a condition is true where n is the first occurrence to be reported and m is the last occurrence to be reported.

Output from TRAP consists of the address referenced and the type of reference (read or write).

Output from TRAP is written on the list output dataset for the simulator.

Format:

TRAP,M=*mode*,P1=p_1,P2=p_2,RF=*rw*,A1=*fwa*,A2=*lwa*,$\left\{\begin{matrix}\text{N1}=n_1,\text{N2}=n_2\\ \text{E1}=e_1,\text{E2}=e_2\end{matrix}\right\}$,I=$i$,EXIT,
C=$cdx_1cj_1cdx_2cj_2\ldots cdx_n$,ON=*on*,OFF=*off*,TAG=$t$.

M=*mode* — CPU mode:

- E EXEC (monitor)
- S STP
- U User (default)
- Z Trap regardless of state (relative to current base address)

P1=*p*
P2=*p* — Parcel address of instruction at which trap is to occur; a word address suffixed by A, B, C, or D. If P is present the trap is conditional and I and C have meaning.

RF=*rw* — Type of reference:

- R Read
- W Write
- RW Read or write (default)

A1=*fwa* — Absolute word address specifying lower bound of trap range relative to base address

A2=*lwa* Absolute word address specifying upper bound of trap range relative to base address

N1=*n*
E1=*e* See section 6.4.2. Defines the first time to trap on this condition. The default is 1 if the keyword is specified without a value.

N2=*n*
E2=*e* See section 6.4.2. Defines the last time to trap on this condition. The default is 2^{48}-1 if no value is given.

I=*i* Increment. The trap occurs every i^{th} time this condition is encountered. The default increment is 1.

EXIT† If EXIT is specified, the simulator looks for an EXIT directive after generating the trap output.

C=*cdx*$_1$*cj*$_1$*cdx*$_2$*cj*$_2$...*cdx*$_n$†

This optional parameter defines condition(s) for a trap. Two or more conditions may be connected by a conjunction. Conditions are evaluated from left to right. A trap is taken every i^{th} time the condition(s) are true.

cdx$_i$ A condition has the following form:

$$\left\{\begin{array}{l} reg \\ (locn) \end{array}\right\} \; op \; \left\{\begin{array}{l} reg \\ (locn) \\ value \end{array}\right\}$$

reg Register designator: An, Bn, Sn, Tn, or Vnee. For An, Sn, or Vnee, n is in the range 0 through 7. For Bn and Tn, n is in the range 0 through 77. ee is a 2-digit number in the range 00 through 77.

(locn) Designates the contents of a given location.

value An octal, decimal, or ASCII value. A floating-point value is assumed if D'*value* contains a decimal point.

† Deferred implementation

op One of the following operators:

op	Significance
>	Greater than
<	Less than
=	Equal to
[	Less than or equal to
]	Greater than or equal to
#	Not equal to

cj$_i$ A conjunction joins two conditions and is one of the following:

conj	Operation (where A and B are conditions)
&	Logical AND of A and B
!	Logical OR of A and B
\	Exclusive OR of A and B
%	A AND NOT B
?	A OR NOT B
/	(A AND NOT B) OR (B AND NOT A)

ON=*on* Turns on each directive having TAG=ON. See section 6.4.2.

OFF=*off* Turns off each directive having TAG=ON. See section 6.4.2.

TAG=*t* Assigns a numeric label to provide control by an ON or OFF parameter. See section 6.4.2.

Example:

```
TRAP,M=U,A1=200,A2=300,N1=1,N2=25,I=5,C=S2=0&S3=1.
```

6.6 SIMULATED OPERATOR STATION COMMANDS

Operator commands including dataset staging can be issued from two channels (1 and 12). Timing control for issuing these commands is also provided.

6.6.1 OPERATOR COMMAND FORMAT

The STATION commands that are currently implemented and their formats are:

DEVICE, *device*, {ON / OFF} ;
DROP, *jsq*;
JOB, *jobname* {,*jsq* } ;
KILL, *jsq*;
LIMIT, *n*;
LINK;
LOGOFF;
LOGON {,*id* }{,'*tid*'} ;
OPERATOR, *nid*, '*tid*';
RERUN, *jsq*;
RESUME, *jsq*;
ROUTE, *oid*, *nid*;
SAVE, *dataset*;
STATUS;
STORAGE;
STREAM, *id*, *no*, *ni*, *na*;
SUBMIT, *filename*;
SUSPEND, *jsq*;

Note that no abbreviations are supported.

6.6.2 CHANNEL CONTROL DIRECTIVE

The simulator is initialized with all operator command streams assigned to channel 1. To switch channels, the channel control directive is inserted in the operator station command directives. Only channel 12 is supported as an alternate channel by the simulator.

Format

CHANNEL *n*;

where *n* may be 1 or 12

Example:

```
LIMIT,0;
CHANNEL,12,ON;
CHANNEL,12;
SAVE,TESTJOB;
CHANNEL,1
LIMIT,1;
```

6.6.3 OPERATOR COMMAND DELAY COUNT

Each operator command can be delayed relative to the previous command by a delay count. Each unit in the count is approximately equivalent to the simulated execution of 50,000 instructions. When the simulator detects that the operating system is executing in the idle loop, the next operator command is processed immediately. The delay count field is an optional field preceding any operator command.

{*delay count*} *command*;

Example:

```
LIMIT,0
SUBMIT,JOBA;
SUBMIT,JOBB;
2000 LIMIT,2;      --If the jobs are small, this takes
                   --effect when the simulator encounters
                   --the idle loop.
10  KILL,2;        --Approximately 500,000 instructions
                   --later, JOB B will be killed.
```

6.7 SAMPLE CSIM JOB

```
JOB,JN=JSIM,M=1600,T=205.          Job for running simulator.
      .
      .
      .
ACCESS,DN=TESTCOS.                 Access COS dataset.
ACCESS,DN=TESTPAR.                 Access dataset containing parameter file.
COPYF,I=$IN,O=TESTJOB,NF=2.        COPY simulated job and its data to TESTJOB.

SKOL,M=CSIMMAC.                    Invoke the station-command precompiler.
CFT,L=O,E=4,I=$SKF.                Compile SKOL output.
LDR,DN=CSIMR:$BLD,NX,AB=CSIM.      Loads the CSIM relocatable binary.
CSIM,LINES=1200,T=200,OPSYS=TESTCOS,OSPAR=TESTPAR.
                                   Call simulator; OS on TESTCOS, parameters
                                   on TESTPAR.

COPYD,I=JOB,O=$OUT.                Copy output from simulated job to $OUT.
EXIT.
COPYD,I=JOB,O=$OUT.
<eof>
JOB,JN=JOB.                        Job copied to TESTJOB.
COPYF,I=$IN,O=$OUT.
      .
      .
      .
<eof>
      .                            Data for simulated job copied to TESTJOB.
      .
      .
<eof>                              Station directives.
STATION COMMANDS:                  Begin SKOL input (required).
SAVE,COPYF;                        Simulate the staging of COPYF.
10 SUBMIT,TESTJOB;                 Test delay-time units later, stage.
```

```
END COMMANDS;                    End SKOL input (required).
<eof>                            Simulator directives (station directives
                                     must be first).
  DS.                            Deadstart simulated system.
  START.
  EXIT.
```

EXTRACT 7

7.1 INTRODUCTION

EXTRACT is an operating system utility program for selectively extracting and processing the contents of a system logfile containing messages issued by users and tasks during normal system operation.

The system logfile is maintained in the Message Processor (MSG) task and contains a record for each message. The format and types of messages are described in part 1, section 3.10.

EXTRACT is loaded into the user field and executed as a user job step when an EXTRACT control statement is encountered. User control of the EXTRACT program is through a set of directives comprising the next file on the $IN dataset for the job. EXTRACT scans a system logfile, which may be the current running logfile, looking for records that satisfy user-specified criteria. Upon finding a record, it processes it according to record type and subtype; for example, it may calculate a CPU usage charge and write an entry on a report.

Internal design of EXTRACT is described in Part 1, section 10.

7.2 EXTRACT STATEMENT

The EXTRACT statement consists simply of the verb EXTRACT: it has no accompanying parameters:

```
EXTRACT.
```

7.3 DIRECTIVES

Directives permit extensive user selection of the data to be extracted from the system logfile and control over the format in which this data is to be presented; that is, how the data is to be processed.

7.3.1 GENERAL FORMAT OF DIRECTIVES

A directive has the following general form:

```
verb = parameter,
```

The = is optional and is allowed for improved readability.

A directive may begin in any column of a card but must be wholly contained on a given card; there is no means of continuing a directive to the next card. A single card may contain multiple directives separated by commas or semicolons. The final directive on a card need not be terminated since the end of the card implies a comma or semicolon. Blanks occurring in directives are optional and are ignored.

The following directive verbs are supported by EXTRACT:

SELECT	DUMP
INPUT	RAWDUMP
OUTPUT	NOHEADER
LINES	LEFT8
FLUSH	RIGHT8
END	

These are described in detail in later subsections. A directive statement consists of a directive verb and sometimes a keyword and a value according to the requirements of the verb. In the special case of SELECT SUBTYPE, there may be multiple values.

When the verb is omitted from a directive, it is assumed to be SELECT. For each directive, a value is set in a table for use by EXTRACT during execution. If a directive is omitted from the set, EXTRACT uses a default value in its place. For example, the OUTPUT directive specifies the dataset on which the report is written. Its default is the $OUT dataset. Thus, the OUTPUT directive is required only when a user wants the EXTRACT output written on a dataset other than $OUT.

A null or empty directives file causes EXTRACT to dump all entries in the system logfile.

The END directive permits multiple sets of EXTRACT directives on a single file.

7.3.2 SELECT DIRECTIVE

SELECT directives specify records to be selected for processing. Multiple SELECT directives may occur in a set of directives.

Formats of the SELECT directive are:

SELECT *field=value,*

and *field=value,*

where *field* refers to a field in the message record accessible to EXTRACT.

Because some fields used for selection are contained in only certain message types, not all selection criteria are relevant for all message types. The description of the system logfile given in section 3.10 indicates which fields are relevant for specific types.

field=value	Description	
TYPE=*type*	Specifies major type of log entry where type can be:	
	ACCOUNT	Selects all records pertaining to machine use by a user job; see section 7.5.
	ASCII	Selects ASCII character string messages; this category includes all user-oriented messages
	HARDWARE	Selects all hardware errors detected by the operating system during normal production
	MISC†	Selects miscellaneous records such as attempted security violations
	SPM	Generates performance and usage reports on COS, the central processor, and system I/O.
	STARTUP	Selects messages issued by system Startup
	STATION	Selects station-related messages

† Deferred implementation

field=value	Description

{SUBTYPE / SUB} = *subtype*$_1$ *subtype*$_2$ ... *subtype*$_n$

Selects subtypes relevant to a particular major type. Multiple subtypes, up to a maximum of 5, can be specified on a single SELECT directive, for example:

```
SELECT  SUBTYPE = JSH PDM USER
```

ACCOUNT subtypes

INIT†	Job initialization message written when job is assigned JXT entry
TERM	Job termination message
OPEN†	Open dataset messages
CLOSE†	Close dataset messages
DISPOSE†	Messages indicating that a job has disposed a dataset

ASCII subtypes

DQM	Disk Queue Manager messages
PDM	Permanent Dataset Manager messages
MSG / LOG	Message Processor messages
JSH	Job Scheduler messages
USER	User-originated messages
CSP	Control Statement Processor messages
EXP	Exchange Package Processor messages
SCP	Station Call Processor messages
STP	Task initialization messages
ABORT	User abort messages
MEP	Memory Error Processor messages
DEC	Disk Error Correction messages
SPM	System Performance Monitor messages

HARDWARE subtypes

DISK	Disk errors
CHANNEL†	Channel errors
SINGLE	Corrected memory errors
DOUBLE	Uncorrected memory errors

† Deferred implementation

field=value	Description
	SPM subtypes
	CPU — CPU usage
	TASK — Task usage
	EXECREQ — Executive requests from each task
	MEM† — User memory usage
	DISK† — Disk usage
	DISKCHAN — Disk channel usage
	LINK† — Link usage
	EXECCALL — Executive call usage
	USERCALL† — User call usage
	CHANINT — Channel interrupt counts
	JSHSTAT — Job Scheduler statistics
	CLASS — Job class statistics
	STARTUP subtypes
	PDR — Permanent dataset recovery messages
	RRJ — Recovery of rolled job messages
	STATION subtypes
	RECEIVED — Messages relating to staging in of datasets
	TRANSMIT — Messages relating to staging out of datasets
JOBNAME=*jn*	Specifies that only records pertaining to the named job are to be extracted.
USER=*n*†	Specifies that only records pertaining to the specified 15-character user number are to be extracted.
DATASET=dn †	Specifies that only records pertaining to the specified 7-character dataset name are to be extracted.

† Deferred implementation

STARTIME=*hours*:*minutes*:*seconds* *month*/*day*/*year*

Specifies the time and day after which messages will be scanned in the logfile. Each value must be given as two digits. The time of day cannot be omitted from this directive. The date can be omitted and is then assumed to be today's date.

Example:

STARTIME=14:28:00 03/14/77

If no STARTIME directive is provided, EXTRACT scans from the beginning of the logfile.

field=value	Description
ENDTIME=*hours*:*minutes*:*seconds* *month*/*day*/*year*	Specifies the time and day at which message scanning of the logfile terminates. Each value must be given as two digits. The time of day cannot be omitted from this form of the directive. The date can be omitted and is assumed to be today's date. If no ENDTIME directive is provided, scanning of the logfile terminates at the end of the logfile. If the current logfile is being scanned, this means all messages up to the present time are reported.
ENDTIME=NOW	This special form of the ENDTIME field causes the date and time to be set to the time of the EXTRACT run. This form of the field can be used when there are future times in the system log.
{JOBSEQ / JSQ}=*n*	Specifies that only records pertaining to the indicated sequence number of a user job are to be processed.
SOURCE=*sid*†	Selects messages only for jobs or datasets originating at front-end computer identified by station id.
DEST=*sid*†	Selects messages for jobs or datasets defined for front-end computer identified by station id.
$LOG={YES / NO}	If YES, selects messages that are also written to a user log. If NO, user log messages are not selected. The default is all messages, whether written to a user log or not. This is the same as YES and NO combined.
TERMINAL=*tid*†	Selects messages for jobs or datasets destined or originating at terminal specified by terminal id.
MSGID=*msgid*	Selects ASCII type messages that are prefixed by the specified 5-character identifier, for example, MSGID=FT001.

† Deferred implementation.

7.3.3 INPUT DIRECTIVE

The INPUT directive specifies the local dataset name of the system logfile to be scanned. This directive allows several logfiles to be processed in one job. The INPUT directive is not required if only one logfile dataset is to be processed and if that dataset has been accessed with the local dataset name of $SYSLOG.

Format:

INPUT = *dn*,

Example:

INPUT = DAYLOG,

7.3.4 OUTPUT DIRECTIVE

The OUTPUT directive specifies the name of the dataset on which the EXTRACT report is to be written. The default dataset is $OUT.

Format:

OUTPUT = *dn*,

Example:

OUTPUT = TEMP,

EXTRACT output is written on TEMP rather than $OUT.

7.3.5 LINES DIRECTIVE

The LINES directive specifies the maximum number of records that EXTRACT will write on the output dataset.

Format:

LINES = *n*,

where n is a decimal line count. The default line limit is 1000.

Example:

LINES = 500,

7.3.6 FLUSH DIRECTIVE

The FLUSH directive requests EXTRACT to write 100 dummy messages in the system log. This causes the system log memory buffers to be flushed to mass storage.

Format:

```
FLUSH,
```

7.3.7 NOHEADER DIRECTIVE

The NOHEADER directive turns off page headings in the EXTRACT report. This feature is useful when the report is to be processed by some other program.

Format:

```
NOHEADER,
```

Example:

```
MSGID=FT001,NOHEADER,LINES=10000.
```

7.3.8 DUMP DIRECTIVE

In addition to the normal report, DUMP causes EXTRACT to write the report in octal formatted into 64-bit words.

Format:

```
DUMP,
```

7.3.9 RAWDUMP DIRECTIVE

The RAWDUMP directive causes EXTRACT to write the raw source data from which the report was taken onto a dataset named RAWDUMP.

Format:

```
RAWDUMP,
```

7.3.10 LEFT8 AND RIGHT8 DIRECTIVES

The LEFT8 and RIGHT8 directives are designed for those installations with a CRAY-1 hardware configuration that has either the left or the right 8 banks of memory. These directives are used in directive sets that specify hardware type messages with memory error subtype(s). They cause the addresses in the memory failure records to be modified to reflect the design of the particular 8-bank configurations and should be useful to engineers at these sites.

Formats:

```
LEFT8,
  or
RIGHT8,
```

Example:

```
TYPE = HARDWARE
SUBTYPE = SINGLE,LEFT8
```

7.3.11 END DIRECTIVE

The END directive (or simply a period) allows multiple directive sets to be processed in one EXTRACT run.

Formats:

```
{ END }
{ .   }
```

Example:

```
SELECT TYPE = ASCII, LINES = 10000,
OUTPUT = DAYLOG, END
SELECT TYPE = HARDWARE.
```

The final directive is optionally terminated by END or period; that is, neither is required.

7.4 DIRECTIVE SETS

A file of directives may contain more than one set of directive. EXTRACT processes directives in a set until it encounters an END statement or a period and then processes the system logfile. For example, the following consists of two sets of directives. The first set uses the system logfile with the local dataset name FIRSTLOG and writes a report on FILEA. The second reads the logfile with the local dataset name $SYSLOG and writes a report on FEFILE.

```
OUTPUT = FILEA, INPUT = FIRSTLOG, SELECT TYPE = ASCII, SELECT SUBTYPE =
PDM, END, OUTPUT = FEFILE, TYPE = HARDWARE, SUBTYPE = DISK
```

7.5 ACCOUNTING RECORD FEATURE

As a special feature, EXTRACT writes an unformatted binary record on a dataset named ACCOUNT each time it processes an accounting record.

An accounting record contains the following items:

- Job name
- User ID
- Source ID
- Terminal ID
- Time executing (in floating point seconds)
- I/O requests (floating point value)
- Blocks transferred
- Execution memory integral (execution time times field length, expressed as megaword seconds)
- I/O memory integral (execution time blocked for I/O expressed in megaword seconds)
- Disk channel time
- User's JOB statement priority
- Time staged in at the CRAY-1 (clock periods)
- Date and time JXT entry made; i.e., initiation time (clock periods)
- Termination time (clock periods)
- Job sequence number
- User number from the JOB statement

7.6 EXAMPLES

(1) The following illustrates the job control statement file and directives file for an EXTRACT run. The directives instruct EXTRACT to print all type ASCII entries in the system logfile pertaining to jobs named SAMS-JOB that ran after noon today.

```
JOB,JN=TESTJOB.
ACCESS,DN=$SYSLOG,PDN=$SYSTEMLOG,R=SECRET.   Access current systemlog.
EXTRACT.
<eof>
     SELECT TYPE = ASCII,
     SELECT JOBNAME = SAMSJOB,
     STARTIME = 12:00:00
<eod>
```

(2) This example of only the directives file causes EXTRACT to print a report of disk errors.

```
     TYPE = HARDWARE, SUBTYPE = DISK
```

(3) The following control statements and directives process $SYSLOG to produce the report shown in figure 7-1.

```
JOB,JN=CRAYLOG,T=60,M=50,P=14.
ACCESS,DN=$SYSLOG,PDN=$SYSTEMLOG.
EXTRACT.
<eof>
     TYPE = ASCII, STARTIME = 09:30:00, LINES = 5,
     TYPE = ACCOUNT, LINES = 3.
     TYPE = STATION, STARTIME = 15:00:00 02/06/80, LINES = 3.
     TYPE = HARDWARE, LINES = 3.
<eod>
```

Note that in this example each line is a separate directive set.

(4) This example of the directives file causes EXTRACT to print a report of all subtypes of SPM records.

```
     TYPE=SPM
```

A portion of this report is shown in figure 7-2.

```
        EXTRACT VERSION X.08     COMPILED 02/06/80  RUN 02/07/80  14:53:03   ON SYSTEMLOG 02/07/80          PAGE   1

09:30:00.7251  USER       86 NAGVER1  LD010 - BEGIN EXECUTION
09:30:00.7286  USER       86 NAGVER1     STOP                IN $MAIN
09:30:00.7295   CSP       86 NAGVER1  ABORT.                                                                 CT406.835
09:30:00.7301 ABORT       86 NAGVER1  AB021 - DATASET NOT FOUND
09:30:00.7304   EXP       86 NAGVER1        - DSNAME: ABORT

- - - - - - END OF EXTRACT REPORT          48214 RECORDS READ FROM $SYSLOG              5 RECORDS WRITTEN ON $OUT


    02/06/80

14:01:06.0994  ACCOUNT   144 T29      USERID =           USERNO =                  SID = MG  TID = :LD2:LJL
                                     126.987 CP-SEC    14378. IOB       12.31777 MEGAWORD-SEC
14:04:29.8782  ACCOUNT   148 TCFT     USERID =           USERNO = COSTEST          SID = DG  TID = OPERATOR
                                       0.424 CP-SEC      186. IOB        0.06761 MEGAWORD-SEC
14:04:56.1508  ACCOUNT   149 CT406    USERID =           USERNO = FOREMAN          SID = DG  TID = OPERATOR
                                      19.229 CP-SEC     2846. IOB        1.23769 MEGAWORD-SEC

- - - - - - END OF EXTRACT REPORT           4329 RECORDS READ FROM $SYSLOG              3 RECORDS WRITTEN ON $OUT

15:03:46.4306 RECEIVED   297 H2109T          SZ = 00000001000   SID = DG   DID = DG
                                     TID = OPERATOR   USERID =
15:03:47.0460 RECEIVED   298 H2109T          SZ = 00000001000   SID = DG   DID = DG
                                     TID = OPERATOR   USERID =
15:07:46.9807 RECEIVED   299 UI003           SZ = 00000002000   SID = DG   DID = DG
                                     TID = OPERATOR   USERID =

- - - - - - END OF EXTRACT REPORT          13116 RECORDS READ FROM $SYSLOG              3 RECORDS WRITTEN ON $OUT

17:26:57.3783   SINGLE  BIT MEMORY ERROR   EM = 1  RM = 1  SYN = 260  ADDR = 01075236  BIT =  8  LOCATION = L-16 W21
                ERROR WORD = 0400010000026001075236  JOBNAME = STP       BA = 00000000  P = 00013547A

    02/07/80

10:56:23.2171   SINGLE  BIT MEMORY ERROR   EM = 1  RM = 0  SYN = 100  ADDR = 00034025  BIT = 70  LOCATION = 0- 6 W00
                ERROR WORD = 0400000000010000034025  JOBNAME = STP       BA = 00021000  P = 000634134
13:41:09.8723   SINGLE  BIT MEMORY ERROR   EM = 1  RM = 3  SYN = 147  ADDR = 01377752  BIT = 38  LOCATION = K-46 W27
                ERROR WORD = 0400030000014701377752  JOBNAME = CT407     BA = 01251000  P = 00126741B

- - - - - - END OF EXTRACT REPORT          81699 RECORDS READ FROM $SYSLOG              3 RECORDS WRITTEN ON $OUT
```

Figure 7-1. Example of EXTRACT report

```
11:12:02.9002        CHANNEL INTERRUPT COUNTS          TIME INTERVAL =    60012.18 MILLISECONDS
                       CHANNEL  0 =          0          CHANNEL 17 =          0
                       CHANNEL  1 =          0          CHANNEL 18 =          0
                       CHANNEL  2 =        118          CHANNEL 19 =          0
                       CHANNEL  3 =        116          CHANNEL 20 =          0
                       CHANNEL  4 =        611          CHANNEL 21 =          0
                       CHANNEL  5 =        604          CHANNEL 22 =          0
                       CHANNEL  6 =       1819          CHANNEL 23 =          0
                       CHANNEL  7 =       1090          CHANNEL 24 =          0
                       CHANNEL  8 =          0          CHANNEL 25 =          0
                       CHANNEL  9 =          0          CHANNEL 26 =       5627
                       CHANNEL 10 =          0          CHANNEL 27 =          0
                       CHANNEL 11 =          0          CHANNEL 28 =          0
                       CHANNEL 12 =          0          CHANNEL 29 =          0
                       CHANNEL 13 =          0          CHANNEL 30 =        496
                       CHANNEL 14 =          0          CHANNEL 31 =          0
                       CHANNEL 15 =          0          CHANNEL 32 =          0
                       CHANNEL 16 =          0          CHANNEL 33 =          0

11:12:02.9090        EXEC CALL USAGE                   TIME INTERVAL =    60012.18 MILLISECONDS
                       NUMBER OF TYPE  0 CALLS =          0          NUMBER OF TYPE 17 CALLS =          0
                       NUMBER OF TYPE  1 CALLS =          0          NUMBER OF TYPE 18 CALLS =          0
                       NUMBER OF TYPE  2 CALLS =        905          NUMBER OF TYPE 19 CALLS =          0
                       NUMBER OF TYPE  3 CALLS =       1172          NUMBER OF TYPE 20 CALLS =          0
                       NUMBER OF TYPE  4 CALLS =          0          NUMBER OF TYPE 21 CALLS =          0
                       NUMBER OF TYPE  5 CALLS =        104          NUMBER OF TYPE 22 CALLS =          0
                       NUMBER OF TYPE  6 CALLS =        370          NUMBER OF TYPE 23 CALLS =          0
                       NUMBER OF TYPE  7 CALLS =          0          NUMBER OF TYPE 24 CALLS =          0
                       NUMBER OF TYPE  8 CALLS =         24          NUMBER OF TYPE 25 CALLS =          0
                       NUMBER OF TYPE  9 CALLS =        176          NUMBER OF TYPE 26 CALLS =          0
                       NUMBER OF TYPE 10 CALLS =         24          NUMBER OF TYPE 27 CALLS =          0
                       NUMBER OF TYPE 11 CALLS =          0          NUMBER OF TYPE 28 CALLS =          1
                       NUMBER OF TYPE 12 CALLS =          0          NUMBER OF TYPE 29 CALLS =          1
                       NUMBER OF TYPE 13 CALLS =        102          NUMBER OF TYPE 30 CALLS =          1
                       NUMBER OF TYPE 14 CALLS =        307          NUMBER OF TYPE 31 CALLS =          1
                       NUMBER OF TYPE 15 CALLS =        307          NUMBER OF TYPE 32 CALLS =          1
                       NUMBER OF TYPE 16 CALLS =       1997

11:12:02.9090        DISK CHANNEL UTILIZATION          TIME INTERVAL =    60012.18 MILLISECONDS
                                CHANNEL  0 =          0.00 MSEC     0.00%
                                CHANNEL  1 =          0.00 MSEC     0.00%
                                CHANNEL  2 =       2559.79 MSEC     4.27%
                                CHANNEL  3 =       1436.73 MSEC     2.39%
                                CHANNEL  4 =          0.00 MSEC     0.00%
                                CHANNEL  5 =          0.00 MSEC     0.00%
                                CHANNEL  6 =          0.00 MSEC     0.00%
                                CHANNEL  7 =          0.00 MSEC     0.00%
                                CHANNEL  8 =          0.00 MSEC     0.00%
                                CHANNEL  9 =          0.00 MSEC     0.00%
                                CHANNEL 10 =          0.00 MSEC     0.00%
                                CHANNEL 11 =          0.00 MSEC     0.00%
```

Figure 7-2. Listing of all subtypes of SPM records

```
11:12:02.9094       EXECUTIVE REQUESTS                TIME INTERVAL =    60012.18 MILLISECONDS
                          NUMBER OF TASK  0 REQUESTS =          0
                          NUMBER OF TASK  1 REQUESTS =        313
                          NUMBER OF TASK  2 REQUESTS =       1370
                          NUMBER OF TASK  3 REQUESTS =        266
                          NUMBER OF TASK  4 REQUESTS =          0
                          NUMBER OF TASK  5 REQUESTS =        715
                          NUMBER OF TASK  6 REQUESTS =        578
                          NUMBER OF TASK  7 REQUESTS =          0
                          NUMBER OF TASK  8 REQUESTS =         21
                          NUMBER OF TASK  9 REQUESTS =       2238
                          NUMBER OF TASK 10 REQUESTS =          0


11:12:02.9096       TASK UTILIZATION                  TIME INTERVAL =    60012.18 MILLISECONDS
                          NUMBER OF TASK  0 READIES =          0
                          NUMBER OF TASK  1 READIES =        129
                          NUMBER OF TASK  2 READIES =        488
                          NUMBER OF TASK  3 READIES =         86
                          NUMBER OF TASK  4 READIES =          0
                          NUMBER OF TASK  5 READIES =        297
                          NUMBER OF TASK  6 READIES =        156
                          NUMBER OF TASK  7 READIES =          0
                          NUMBER OF TASK  8 READIES =          3
                          NUMBER OF TASK  9 READIES =        410
                          NUMBER OF TASK 10 READIES =          0


11:12:02.9099       CPU UTILIZATION                   TIME INTERVAL =    60012.18 MILLISECONDS
                          USER TIME =       60134.40 MSEC    100.20%
                          IDLE TIME =           0.00 MSEC      0.00%
                          BLOCKED TIME =        0.00 MSEC      0.00%
                          EXEC TIME =         151.90 MSEC      0.25%
                          TASK  0 TIME =        0.00 MSEC      0.00%
                          TASK  1 TIME =       16.42 MSEC      0.03%
                          TASK  2 TIME =       25.84 MSEC      0.04%
                          TASK  3 TIME =        4.39 MSEC      0.01%
                          TASK  4 TIME =        0.00 MSEC      0.00%
                          TASK  5 TIME =       14.85 MSEC      0.02%
                          TASK  6 TIME =       17.61 MSEC      0.03%
                          TASK  7 TIME =        0.00 MSEC      0.00%
                          TASK  8 TIME =        0.41 MSEC      0.00%
                          TASK  9 TIME =       42.38 MSEC      0.07%
                          TASK 10 TIME =        0.00 MSEC      0.00%
                          TIME NOT ACCOUNTED FOR =     -402.02 MSEC     -0.67%

- - - - - - END OF EXTRACT REPORT          43892 RECORDS READ FROM $SYSLOG          49 RECORDS WRITTEN ON $OUT
```

Figure 7-2. Listing of all subtypes of SPM records (cont.)

SPECIAL CONTROL STATEMENT PARAMETERS 8

8.1 INTRODUCTION

Some COS control statements have parameters designed for the use of system analysts debugging or maintaining the operating system. These parameters are not for general use and are thus described in this publication.

8.2 BREAKPOINT PARAMETER

A breakpoint parameter can be used on the JOB control statement and the LDR control statement in addition to other parameters described in the COS reference manual.

Formats:

```
JOB,...,BP,...

LDR,...,BP,...
```

The BP parameter is a system debugging aid. It causes the loader to suspend the loaded program when it is ready for execution before its first instruction is executed. This allows the operator to enter breakpoints before starting program execution. The RUN command must be issued from the station to initiate execution. For the JOB control statement, BP affects every program loaded as a result of being named as a verb on a control statement. For the LDR control statement, only the current load is affected; the parameter does not affect object modules created by use of the AB parameter.

8.2 SDR ENTRY PARAMETER

Specifying an ENTER parameter on an ACCESS or an ACQUIRE control statement causes an entry to be made for a dataset in the System Directory (SDR). Users who subsequently want to load and execute this entered dataset need not use an ACCESS or ACQUIRE prior to using the name of

the dataset as a control statement verb. The dataset being entered into the SDR must contain an absolute binary object module. Datasets usually entered into the SDR include those containing CAL, CFT, UPDATE, BUILD, CSIM, and copy and dump utilities.

Formats:

```
ACCESS,..,ENTER,...

ACQUIRE,..,ENTER,...
```

FDUMP 9

9.1 INTRODUCTION

FDUMP is an operating system utility program for formatting and printing the contents of a dataset containing an image of CRAY-1 memory, according to user-supplied directives. Normally, the dataset is one created by system Startup following a system failure and operator-initiated memory dump. However, any dataset that is properly formatted is acceptable to FDUMP; see section 9.4 for a description of the acceptable dataset format.

FDUMP executes as a job-step within a normal user job during normal system operation. Ordinarily, the dump to be formatted is accessed by the job prior to calling FDUMP. However, the $DUMP dataset written by DUMPJOB can also be used as input to FDUMP. Options available in FDUMP include the ability to specify dumping absolute memory, to copy the dump to another dataset in compressed format, or to decompress the data from a compressed dataset.

9.2 FDUMP CONTROL STATEMENT

The control statement that calls FDUMP into execution is:

```
FDUMP,I=idn,L=odn.
```

Optional parameters:

I=*idn* Specifies name of dataset from which to read user directives. The default is I=$IN. Specifying I with no value also defaults to $IN.

L=*odn* Specifies name of dataset to which all listable output will be written. If L is omitted or has no value specified, output is written to $OUT.

A dataset named on the control statement cannot legally be named on directives within the input dataset.

9.3 DIRECTIVES

FDUMP interprets directives and performs the requested function. All directives are processed sequentially in the order encountered. The format of a directive is similar to that for COS control statements. A directive is a free-form card image consisting of a verb and zero or more optional parameters. Some verbs require parameters. A parameter consists of a keyword, usually followed by a value. Parameters are order-independent. A period terminates the directive. If there is no period before column 80, the end-of-card is treated as a period. Blanks are ignored. The listable output dataset contains a copy of all directives encountered on the input dataset.

FDUMP recognizes two classes of directives. The first class controls the memory to be dumped, specifying which datasets are involved and memory limits to dump. The second class controls listing spacing and titles.

9.3.1 CLASS 1 DIRECTIVES

Class 1 directives include the following:

FILES	Identify datasets containing the current dump and any symbol table datasets to be read in. Multiple dumps can be processed in a single execution by using multiple FILES directives.
DMEM	Format and print memory range. A FILES directive must precede the first DMEM directive.
COMP	Compress dump or portions of dump onto an alternate dataset.
XCOMP	Decompress a dataset created by COMP onto an alternate dataset.
DSYM	Locate and print contents of symbolic location.
AUTO	Dump memory according to a standard set of directives.
DSDT	Format and print the System Dataset Table area from a COS system dump.
DXTR	Format and print the EXEC trace area from a COS system dump.
SETBIAS	Define offset for subsequent addresses.

FILES directive

The FILES directive identifies the datasets containing the dump to be processed and any symbol table datasets to be used for definition of symbolic address or length specifications.

The format of the FILES directive is:

FILES,DDS=*dn*[,SYM1=*symdn*$_1$][,SYM2=*symdn*$_2$][,SYM3=*symdn*$_3$].

Parameters:

DDS=*dn*	Specifies the current dump dataset to be on *dn*. This parameter is required and must not name a dataset specified on the FDUMP control statement.
SYM$_n$=*symdn*$_n$	Specifies optional symbol table datasets. FDUMP will read in the symbol tables from these datasets. They must contain symbol tables in the format written by the CAL assembler.

The dump dataset must be an unblocked dataset and must contain a memory dump in the format written by system startup. It may be a permanent dataset previously accessed by the job. The symbol table datasets, if specified, must be blocked datasets. Symbol table processing includes removal of duplicate symbols from the tables if the duplicated symbol contains the @ character in either the second or third character position. This allows field definition symbols to be removed from the second and third tables, if their values are already known. Other duplicate symbols are retained.

Symbol table searching is ordinarily performed in a priority order, with the dataset identified by SYM1 being searched before SYM2, and SYM2 being searched before SYM3. On most directives supporting symbolic addresses, it is possible to limit the symbol table search to those entries coming from one specific dataset. If FDUMP cannot find a symbolic name in the specified symbol tables, it writes an informative diagnostic to the list dataset and skips the directive.

Once a symbol table has been read in, all symbols defined in that table remain defined until a subsequent FILES directive is encountered specifying a different dataset for the same SYM_n parameter. A symbol table can be removed without replacement by specifying SYM_n=0. Subsequent FILES directives that do not specify a particular SYM_n parameter do not affect that SYM_n parameter.

Examples of FILES directives:

(1) FILES,DDS=DUMP1.

Current dump is on dataset DUMP1; no symbolic information is available.

(2) FILES,DDS=DUMP2,SYM1=SYMS.

Current dump is on dataset DUMP2; symbol table information is on SYMS.

(3) FILES,DDS=X,SYM1=SYMS1,SYM2=SYM2.

Current dump is on dataset X; symbol tables are on dataset SYMS1 and SYM2. The table from SYMS1 is searched first. If no matching symbol is found, the table from SYM2 is searched.

FILES,DDS=Y.

Changes current dump dataset to Y. The directive does not affect the use of SYMS1 and SYM2 for symbol definition.

FILES,DDS=Y,SYM1=Ø.

Deletes dataset SYMS1 from consideration in symbol definition, leaving current dump on Y and symbols on SYM2.

DMEM directive

The DMEM directive formats and prints a range of memory. It can be used in conjunction with the AUTO directive to format memory that AUTO would not ordinarily print. Addresses and lengths can be provided symbolically if symbol table information has been provided. At least one FILES directive specifying the current dump dataset must precede the first DMEM directive. FDUMP provides an informative error message if symbolic names are specified for which it cannot find definitions. Optionally, the user may specify that the memory range is to be dumped in exchange package format.

Formats of the DMEM directive:

```
DMEM,FWA=fwa,LWA=lwa[,LE=len1][,XP][,SDN=dn][,BIAS=bias].
```

```
DMEM,FWA=fwa,L=len2[,LE=len1][,XP][,SDN=dn][,BIAS=bias].
```

Parameters:

FWA=*fwa* Beginning memory address to dump. This may be either an octal value or a symbol defined in the current symbol tables.

LWA=*lwa* Ending memory address to dump. This may be octal or symbolic. If *lwa* is greater than the actual last word address (LWA) in the dump, LWA is used. If LWA is used, L cannot be used.

LE=len_1 Length of an entry in a table. This may be symbolic or octal. If len_1 is not specified, all memory from FWA to LWA is printed as one block. If len_1 is specified, memory from FWA to LWA is dumped in groups of len_1 words. When possible, FDUMP detects unused entries and compresses them to a one line message. This is possible only if an unused entry contains all zeros. This feature is suppressed if XP is specified.

L=len_2 Length of memory block to be dumped. LWA is formed from *fwa* + len_2 - 1. len_2 may be either octal or symbolic. If the L keyword is used, the LWA keyword cannot be used.

XP Keyword specifying formatting of memory as an exchange package. Memory is dumped from FWA to LWA in groups of 16 words. If LWA - FWA + 1 is not a multiple of 16, LWA will be rounded down (never up) to form the nearest multiple of 16. The LE keyword is ignored if XP is used. If LWA - FWA + 1 is not at least 16, FDUMP issues an informative diagnostic and skips the directive.

SDN=*dn* Name of dataset to be searched for symbol definitions. Ordinarily, the dataset searched first is named by a FILES directive as SYM1, followed by SYM2 and SYM3, in order. If SDN=*dn* is specified, only the dataset named *dn* is searched to define symbols. If no FILES directive has named *dn* as a symbol table dataset, FDUMP issues an informative diagnostic and skips the directive.

BIAS=*bias* Biasing value to be added to *fwa* and *lwa* when reading the dump dataset. The addresses *fwa* through *lwa* are listed on the printed dataset, but the locations read from the dump are *fwa+bias* through *lwa+bias*. This allows dumping relocated programs and data with the addresses shown in

listings by specifying bias to be the program load address. This bias value is reset to zero before processing subsequent DMEM directives.

Examples of DMEM directives:

Assume that the following FILES directive is in effect:

```
FILES,DDS=DUMP1,SYM1=EXECSYM,SYM2=STPSYM.
```

where EXECSYM contains the symbol table from assembly of COS EXEC and STPSYM contains symbol table entries from STP.

(1) `DMEM,FWA=Ø,LWA=1ØØØ.`

Dump from absolute address 0 to absolute address 1000 from DUMP1 in 64-bit octal words with ASCII equivalents. Four words are printed per line with first and last lines adjusted so that all other lines begin on a 4-word boundary. Areas where more than one line would be printed with all words containing the same value as the last word previous to the area are compressed to a one-line message.

(2) `DMEM,FWA=SXBF,L=2Ø,XP.`

Locates symbol SXBF in the supplied symbol tables and prints 20_8 words beginning at SXBF, formatted as an exchange package. SXBF is assumed to be relative to location zero of the dump.

(3) `DMEM,FWA=B@STT,L=SZ@STT.`

Dumps memory beginning at B@STT and going for SZ@STT words, in four 64-bit octal words per line.

(4) `DMEM,FWA=SIM,L=1.`

Dumps contents of location SIM in COS EXEC.

(5) `DMEM,FWA=SIM,L=1,SDN=STPSYM,BIAS=XEND.`

Dumps contents of STP location SIM. The SDN parameter forces the STP symbol SIM to be used rather than the EXEC symbol. The BIAS parameter causes the base address of STP to be included when positioning the dump dataset.

(6) `DMEM,FWA=B@SDT,L=SZ@SDT,LE=LE@SDT,BIAS=XEND.`

Dumps the system dataset table area. Each system dataset table entry is dumped as a separate entry.

COMP directive

The COMP directive compresses a dump, deleting portions containing sequences of identical words more than three words long. Each section of memory on the output dataset is preceded by a control word indicating if compression occurred. If compression did not occur, the control word is followed by the uncompressed data. When compression is possible, the control word is followed by one word containing the bit pattern that was identical in all compressed words. The control word specifies first and last word addresses represented by the data following.

Format of the COMP directive:

```
COMP,IDN=idn,ODN=odn[,FWA=fwa,LWA=lwa].
```

Parameters:

IDN=*idn*	Name of dataset containing dump to be compressed
ODN=*odn*	Name of dataset to receive compressed dump image
FWA=*fwa*	Optional beginning address. If *fwa* is present, *lwa* must also be specified.
LWA=*lwa*	Optional ending address. If *lwa* is present, *fwa* must also be specified.

If FWA and LWA are not specified, the entire dump from *idn* is compressed. If FWA and LWA are specified, only those portions of *idn* within the designated range are dumped. Both addresses must be octal values. Multiple COMP directives may occur and may specify the same input and output datasets. The output dataset always contains a dump header (see dump format description, section 9.4). Multiple COMP directives specifying the same output dataset will add the new dumps at the end of the existing dataset.

XCOMP directive

The XCOMP directive decompresses a dump created by a COMP directive so that it can be processed by other dumping directives. The input dataset must contain a dump with a recognizable header. The header must indicate that the following dump is compressed. The output dataset should not exist prior to the XCOMP directive since an existing dataset will be overwritten. All memory contents on the input dataset will be transferred to the output dataset.

Format of the XCOMP directive:

```
XCOMP,IDN=idn,ODN=odn.
```

Parameters:

IDN=*idn*	Name of dataset containing a compressed dump
ODN=*odn*	Name of dataset to receive the decompressed image of the dump from *idn*

Example:

```
XCOMP,IDN=DUMPIN,ODN=DUMPOUT.
```

Transfers the dump from DUMPIN to DUMPOUT in a format suitable for analysis with DMEM and DSYM directives.

DSYM directive

The DSYM directive dumps the contents of one memory location by symbolic name. A FILES directive must precede the first DSYM directive.

Format of the DSYM directive:

```
DSYM,SYM=symbol[,SDN=dn][,BIAS=bias].
```

Parameters:

SYM=*symbol*	Symbolic name of desired memory location. If FDUMP cannot find *symbol* in the symbol tables, it issues a warning message and ignores the directive.
SDN=*dn*	Symbol table dataset to be used to define *symbol*. If *dn* does not match one of the symbol table dataset names in effect from preceding FILES directives, FDUMP issues a warning and ignores the directive.
BIAS=*bias*	Value to be added to the value of *symbol* to locate the word in the dump; may be symbolic or octal. FDUMP issues a warning and ignores the directive if *bias* is symbolic and cannot be found in the symbol tables.

Output from the DSYM directive consists of a block of eight words containing the symbol name, the symbol value, and the contents of the corresponding memory word as a 64-bit octal value with ASCII equivalents. FDUMP places two DSYM outputs on one line by using an internal buffer to construct the line. If a sequence of DSYM directives occurs and the number of DSYM directives in the sequence is not even, a FLUSH directive must be used to print the incomplete line containing the results of the final DSYM directive in the sequence.

AUTO directive

The AUTO directive formats a dump according to a predefined set of directives. In many cases, the dump printed as a result of AUTO is sufficient for analyzing the dump. Sometimes, additional directives will be required to print memory or tables not ordinarily printed by AUTO. A user may insert the predefined set of directives into a buffer in FDUMP during compilation or may read them from a user-specified dataset during FDUMP execution.

Format of the AUTO directive:

```
AUTO[,DN=dn].
```

Parameters:

DN=*dn* Name of optional user-supplied dataset containing directives. If the DN parameter is omitted, the AUTO directive causes FDUMP to read directives from an array generated by the CFT routine AUTODAT. (AUTODAT is a BLOCK DATA subroutine compiled with FDUMP.)

At least one FILES directive must be encountered. It is possible for the directives read as a result of an AUTO directive to contain the FILES directive. Ordinarily, however, the FILES directive would precede the AUTO directive. When the directives from the AUTO processing are exhausted, FDUMP resumes reading directives from the original directive source dataset as named by the I parameter on the FDUMP control statement. Nested AUTO directives may cause unpredictable results.

Examples of the AUTO directive, assuming the following FILES directive has been processed:

```
FILES,DDS=DUMP1,SYM1=EXECSYM,SYM2=STPSYM.
```

(1) AUTO.

Causes a set of directives compiled into BLOCK DATA routine AUTODAT to be executed as if they were being read from the directive input dataset.

(2) AUTO,DN=AUTODIR.

Causes dataset AUTODIR to be used as the source of directives. AUTODIR is not rewound before or after reading. Reading continues to an end-of-file.

DSDT directive

The DSDT directive is a special purpose directive suitable for use only when the dump dataset contains a system dump including STP tables. This directive causes the SDT area to be located and formatted by queue or subqueue for those queues having subqueues. The formatted information produces 1 line of information about each dataset in the queue or subqueue.

Format of the DSDT directive

```
DSDT [,SDN=dn][,RAW][,AVAIL= { NONE } ].
                             { ALL  }
                             { USED }
```

Parameters:

SDN=*dn*	Name of dataset used to locate symbol values needed
RAW	If present, the SDT entry will be dumped in octal after the formatted line
AVAIL=*param*	Controls printing of contents of available queue.

Param	Significance
NONE	No information is printed about any entry in the available queue.
ALL	Information is printed about each entry in the available queue. If an entry has never been used, a one line message appears identifying it as never used. Otherwise, the information in the entry is formatted as if the entry were active in another queue.
USED	Information is printed as for ALL, except that unused entries are skipped.

The DSDT directive requires that symbol table information be provided. If missing symbols are identified, or if the required memory is not present, FDUMP issues a warning and ignores the directive.

The default value for the AVAIL parameter is NONE.

Information in the one line printed for each dataset includes dataset name, permanent dataset name, job sequence number, source and destination ID's.

Parameters:

SDN=*dn*	Name of dataset identifying symbol table dataset to be used to define the symbols for locating the EXEC trace

The DXTR directive requires that the symbols DBF, DBFL, and DBFP be defined, as a minimum. If FDUMP cannot find any of these symbols or if the dump dataset does not contain sufficient memory to locate the trace, FDUMP issues a warning and skips the directive.

Output for the DXTR directive consists of one line per trace entry. This line contains the function code and an identifying mnemonic, the P-register of the calling task, the value of register B0 for the calling task, the contents of XA register from the calling task, the interval in clock periods between this entry and the previous entry, and two words of information that vary according to the reason for the entry. When location DBFP contains a value between zero and DBFL, the trace is dumped from oldest to newest. If DBFP appears invalid, FDUMP dumps the trace from beginning (DBF) to ending (DBF + DBFL - 1) address, and prints a message giving the apparently bad DBFL value.

SETBIAS directive

The SETBIAS directive specifies an offset to be added to all subsequent addresses to locate the corresponding word in the dump dataset. The SETBIAS directive remains in effect until another SETBIAS directive is encountered. When a directive allowing a BIAS keyword is encountered, the bias used for addresses while processing that directive is the BIAS value plus the SETBIAS value currently in effect.

Format of the SETBIAS directive:

```
SETBIAS,BIAS=bval[,SDN=dn][,IND].
```

Parameters:

BIAS=*bval*	New value of global bias. *bval* may be either octal or symbolic. To cancel a previous SETBIAS directive, use *bval*=0.
SDN=*dn*	Name of symbol table dataset currently known to FDUMP to be used to define *bval* if *bval* is symbolic.
IND	Keyword specifying indirect biasing. If IND is not present, the value of *bval*, whether octal or symbolic, is used as the new global bias. If IND is present, the value of *bval* is assumed to be the address of a word within the current dump dataset containing the value to be used as the new global bias. Note that if IND is present, any previous SETBIAS affects the reading of the specified location.

SETBIAS may be used to properly locate and dump memory locations where symbol table values are relative to the origin of a particular program or subroutine rather than relative to word 0 of the field length. When the IND keyword is not present, FDUMP adds the value of *bval* to the current global bias and uses the result as the new global bias. If IND is present, FDUMP adds the present global bias to *bval* to form the absolute memory address of the word containing the new bias. FDUMP also adds the value read from that word to the existing global bias to form the new global bias. The addresses printed on output lines from subsequent dump directives will not include the bias value, although it will be used to locate the specified memory within the dump.

Examples of the SETBIAS directive:

(1)
```
DMEM,FWA=Ø,LWA=17.
SETBIAS,BIAS=2ØØØØ.
DMEM,FWA=Ø,LWA=17.
```

Dumps absolute memory addresses 0 to 17 and absolute addresses 20000 to 20017. Both sets of addresses printed at the left of the page will be 0 to 17.

(2)
```
SETBIAS,BIAS=XEND.
DMEM,FWA=Ø,LWA=5ØØØØ.
```

This sequence dumps memory from a system dump. It results in a dump of absolute memory beginning with the first word of STP and ending with word 50000 of STP.

(3) SETBIAS,BIAS=XEND.
SETBIAS,BIAS=P@JXT,IND.
DMEM,FWA=Ø,L=SZ@JXT,LE=LE@JXT.

Dumps all JXT entries. The address printed for word 0 of the first JXT entry will be zero. The same dump may be obtained by the sequence:

```
SETBIAS,BIAS=XEND.
DMEM,FWA=B@JXT,L=SZ@JXT,LE=LE@JXT.
```

except that the address printed for word 0 of the first JXT entry will be the value of B@JXT rather than zero.

9.3.2 CLASS 2 DIRECTIVES

Class 2 directives include the following:

TITLE	Print a line of text on the listable output
SPACE	Control spacing of listable output
FLUSH	Print incomplete line from DSYM directive

TITLE directive

The TITLE directive writes a line of text on the listable output dataset. The TITLE directive is recognized only while processing an AUTO directive and must contain the characters 'TITLE.' in columns 1-6 and blanks in columns 7 and 8. Beginning with column 9 and continuing to column 80 may be from one to 72 characters of text. This text is printed exactly as it appears. One word containing a double space carriage control character followed by blanks is inserted before the text string. Only characters with ASCII values between 40_8 and 140_8 are allowed; all others will be replaced by blanks.

Examples of the TITLE direcitve:

(1) TITLE. THIS IS A TITLE

(2) TITLE. EXECUTING EXCHANGE PACKAGE

SPACE directive

The SPACE directive inserts blank lines into the listable output. Optionally, it forces a page eject record and page header records to be written to the listable output if fewer than a specified number of lines remain on the page currently being formatted.

Format of the SPACE directive:

```
SPACE[,NL=n][,EJ=m].
```

Parameters:

NL=*n*	Decimal number of blank lines to insert. If keyword NL is specified without a value, no blank lines will be written. If keyword NL is not specified, one blank line will be written.
EJ=*m*	Decimal threshold to force page eject. If keyword EJ is specified, *m* must also be specified. The value of *m* is compared to the number of lines remaining on the page before the *n* blank lines are written. If at least *m* lines remain, no page eject occurs. If fewer lines remain, a page eject and header records are written and no blank line will be written. If the EJ keyword is not specified, *n* blank lines will be written regardless of possible page ejects which may occur when bottom-of-page is reached.

Examples of the SPACE directive:

(1) SPACE,NL=4.

Unconditionally inserts four blank lines. If only one line remains on the page when this directive is encountered, the result is one blank line, page eject, header records, and three blank lines.

(2) SPACE.

Unconditionally inserts one blank line.

(3) SPACE,NL=2,EJ=1Ø.

Inserts two blank lines if at least ten lines remain on page. If fewer than ten lines remain, a page eject and header records are written and no additional blank lines are written.

(4) SPACE,NL,EJ=5.

Causes a page eject if fewer than five lines remain on the current page. If at least five lines remain, no spacing occurs.

(5) SPACE,NL,EJ=1ØØ.

Unconditional page eject (it is not possible for 100 or more lines to remain on the current page).

FLUSH directive

The FLUSH directive, used in conjunction with the DSYM directive, causes an incomplete line buffer (resulting from an odd number of DSYM directives) to be written and cleared. If no incomplete buffer exists, no action occurs.

Format of the FLUSH directive:

```
FLUSH.
```

Examples of the FLUSH directive:

(1)
```
DSYM,SYM=A.
DSYM,SYM=B.
DSYM,SYM=C.
FLUSH.
```

Results in one line of output containing symbolic dumps of symbols A and B followed by one line containing symbol C.

(2)
```
DSYM,SYM=A.
DSYM,SYM=B.
DSYM,SYM=C.
DMEM,FWA=Ø,L=5.
DSYM,SYM=D.
```

Results in one line containing A and B, two lines containing locations 0 through 5, followed by one line containing symbols C and D. If the DSYM,SYM=D. is omitted, the value of symbol C is not printed since no FLUSH directive appears.

(3) DSYM,SYM=A.
DSYM,SYM=B.
FLUSH.

Results in one line containing symbols A and B. The FLUSH directive acts as a no-operation directive since the sequence contains an even number of DSYM directives.

9.4 SYSTEM DUMP FORMAT

The format of a system dump created by system Startup is shown in figure 9-1.

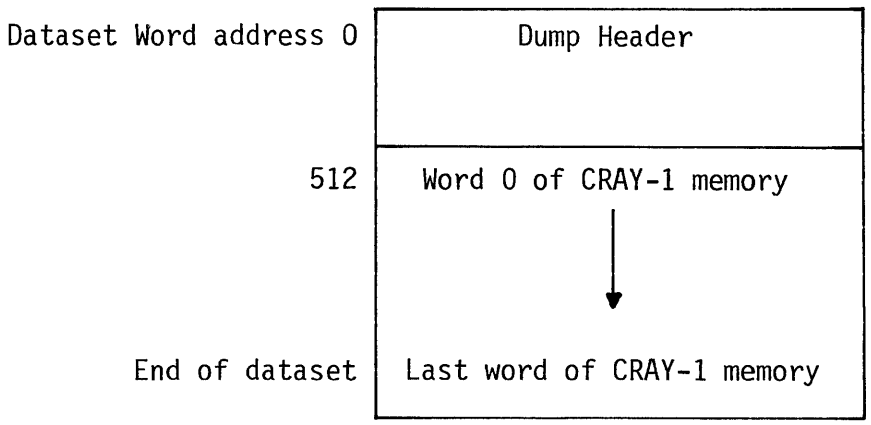


Figure 9-1. Format of a Startup-created dump dataset

The dump header at the beginning of the dataset is 512 words long and has the format shown in figure 9-2.

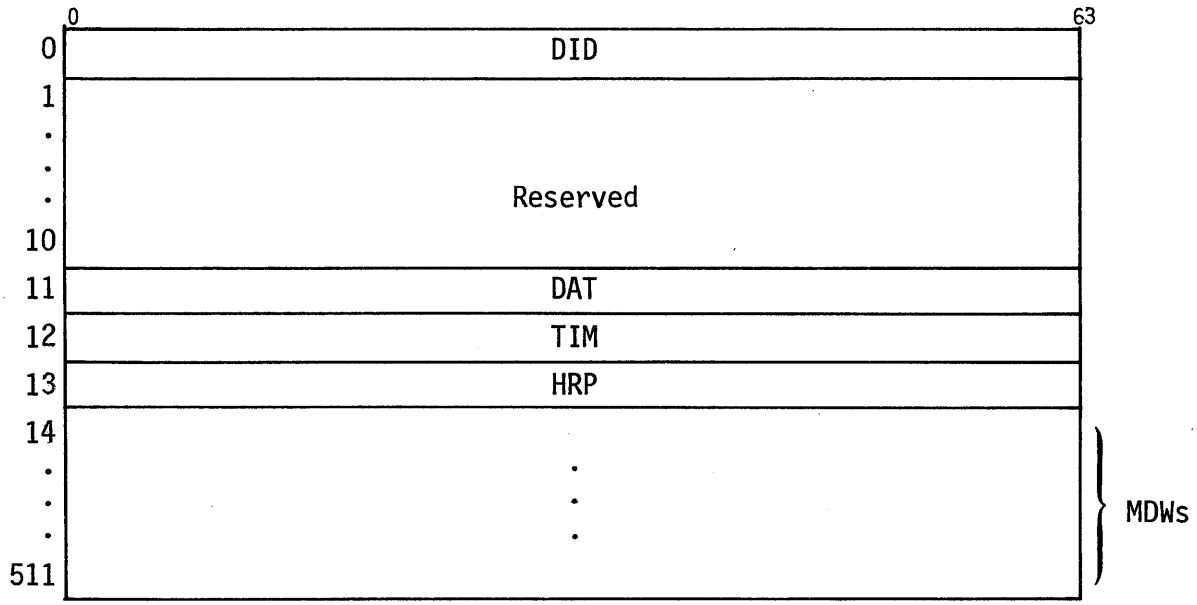


Figure 9-2. Format of a dump header

Field	Word	Bits	Description
DID	0	0-63	Dump identification. Currently set to 'SYSDUMP^' by default. Future implementation will allow operator specification of dump ID.
Reserved	1-10	0-63	Reserved for use as comment field. Currently all blanks by default. Future implementation will allow operator specification of comments.
DAT	11	0-63	Date of Startup. Taken from the MCU during Startup; not necessarily related to the time of the actual system failure that caused the dump to be taken.
TIM	12	0-63	Time of Startup. Taken from the MCU during Startup; not necessarily related to the time of the actual system failure that caused the dump to be taken.
HRP	13	0-63	Header recognition pattern. The default is *DMPHDR*.

Field	Word	Bits	Description
MDW	14-511	0-63	Memory Descriptor Word Pairs (MDWs). Two words per entry, with unused entries zeroed. A dump created by Startup will ordinarily contain only one MDW, which indicates machine field length included in the dump. The format of an MDW is shown in figure 9-3.

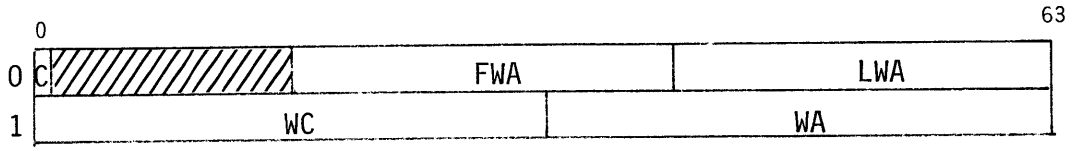


Figure 9-3. Format of a Memory Descriptor Word Pair (MDW)

Field	Word	Bits	Description
C	0	0	Compression flag. Indicates whether the memory represented by this MDW has been compressed.
FWA	0	16-39	First memory word address of this MDW
LWA	0	40-63	Last memory word address of this MDW (inclusive)
WC	1	0-31	Number of words on dataset occupied by memory represented by FWA and LWA. For C=0, WC=LWA-FWA+1.
WA	1	32-63	Word address on dataset of the memory word represented by FWA, relative to beginning of dataset.

A system dump generated by Startup is never compressed. Only a dump created by FDUMP using the COMP directive is compressed. A compressed dump follows the same general format as an uncompressed dump, except that one additional control word called a Dump Control Word (DCW) is used in a compressed dump. The WA field of the MDW for a compressed dump points to the dataset word address of the DCW for the memory word represented by FWA rather than pointing to FWA.

Compression is accomplished by replacing all occurrences of three or more consecutive words containing identical bit patterns with a compression DCW followed by one word indicating the matching bit pattern. If a block of words is encountered for which no compression is possible, the data is indicated by a non-compression DCW, followed by the block of data. This may be only one word if another compressible block begins immediately. The format of a DCW is given in figure 9-4.

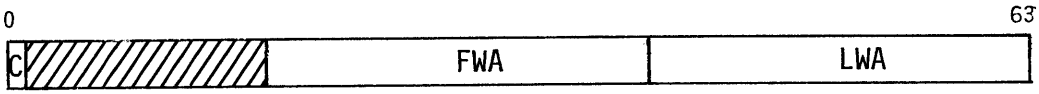


Figure 9-4. Format of a Dump Control Word (DCW)

Field	Bits	Description
C	0	Compression flag. DCW represents compressed data if C≠0.
FWA	16-39	Memory word address of first word following DCW
LWA	40-63	Last word address represented by memory block following DCW

If field C is non-zero, only one data word follows the DCW. This single word represents LWA-FWA+1 occurrences of that bit pattern in consecutive words of memory. If field C is zero, LWA-FWA+1 words of data follow the DCW. Another DCW followed by another block of data may follow the last word of data for the current DCW. The WA field of the MDW corresponding to this group of DCWs and data points to the first DCW in the group.

Multiple COMP directives specifying the same output dataset will create multiple MDWs, each resulting from one COMP directive and each having its own set of DCWs and associated data.

Examples of compressed dump formats:

A dump consisting of the following memory contents will appear as described below, assuming the sequences of directives given in the examples.

Dump contents:

Word	Contents
0	0
1	-1
2-10	0
11-20	15
21	2
22-36	15
37	5

(1) COMP,IDN=DUMP,ODN=CDUMP.

Produces dataset named CDUMP having the following structure:

Word	Contents
0-13	Dump header (See figure 9-2)
14-15	MDW having the following field values (octal): C=1, FWA=0, LWA=37, WC=15, WA=1000.
16-511	0 (unused)
512	DCW having field values (octal): C=0, FWA=0, LWA=1
513	0
514	-1
515	DCW having field values: C=1, FWA=2, LWA=10
516	0
517	DCW having field values: C=1, FWA=11, LWA=20
518	15
519	DCW having field values: C=0, FWA=21, LWA=21
520	2
521	DCW having field values: C=1, FWA=22, LWA=36

Word	Contents
522	15
523	DCW having field values: C=0, FWA=37, LWA=37
524	5
525-1023	0 (unused)
	<eod>

(2) COMP,IDN=DUMP,ODN=CDUMP1,FWA=Ø,LWA=15.
COMP,IDN=DUMP,ODN=CDUMP1,FWA=2Ø,LWA=25.
COMP,IDN=DUMP,ODN=CDUMP1,FWA=27,LWA=37.

produces dataset having the following structure:

Word	Contents
0-13	Header
14-15	MDW having octal field values: C=1, FWA=0, LWA=15, WC=7, WA=1000.
16-17	MDW having octal field values: C=1, FWA=20, LWA=25, WC=5, WA=1007.
18-19	MDW having octal field values: C=1, FWA=27, LWA=37, WC=10, WA=1014
20-511	0 (unused)
512	DCW having octal field values: C=0, FWA=2 ,LWA=1
513	0
514	-1
515	DCW having field values: C=1, FWA=0, LWA=10
516	0
517	DCW having field values: C=1, FWA=11, LWA=15
518	15
519	DCW having field values: C=0, FWA=20, LWA=21
520	15
521	2

Word	Contents
522	DCW having field values: C=1, FWA=22, LWA=25
523	15
524	DCW having field values: C=1, FWA=27, LWA=36
525	15
526	DCW having field values: C=0, FWA=37, LWA=37
527	5

SYSTEM STATISTICS (STATS) 10

10.1 INTRODUCTION

STATS gathers performance statistics for the CRAY-1 in an operating environment. It provides the following information obtained from the $SYSTEMLOG file:

- Daily statistics including:

  Number of jobs
  CPU time
  NUMBER OF I/O blocks
  Link traffic
  Number of deadstarts

- Deadstart log
- Monthly summary of the daily statistics

All of the information is obtained from the $SYSTEMLOG datasets and a summary file called STATSFOR. The program is as fully automated as is currently possible. It has its own internal routines to read the systemlog and is able to gather statistics across systemlog boundaries.

10.2 CONTROL STATEMENT

The STATS routine is called into execution by the STATS control statement. Parameters on the control statement determine starting and stopping times.

The format is:

```
STATS,ED=ed,SDATE=sdate,EDATE=edate,LASTED=ed.
```

Parameters are in keyword form:

ED=*ed* — $SYSTEMLOG edition at which to start gathering statistics. The default is the highest edition number.

SDATE=*sdate* — The date at which to start gathering statistics. Default is the day following the last date in the 'STATSFOR' dataset. If no 'STATSFOR' dataset exists, SDATE is required.

EDATE=*edate* — The date at which to stop gathering statistics. Default is the current date.

LASTED=*ed* — The last edition of $SYSTEMLOG. Normally, STATS stops statistics gathering when it can no longer access the next edition at $SYSTEMLOG. If an edition is missing (for example .... 2,3,4,6,7, & 8), this parameter may be used to force STATS to ignore missing editions until *ed* is processed.

The dates must be of the form mm/dd/yy.

10.3 DATASET REQUIREMENTS

The STATSFOR dataset contains all the information for each day to provide a monthly summary. If the dataset does not exist, the program creates it. At the end of every run, the dataset is updated to include the information from the current run and a copy is disposed to the station to allow recovery of the dataset should it be deleted.

If no ED or SDATE parameters are present on the control statement, the program reads the STATSFOR dataset to get the last data processed. It then calculates the next day's date and looks in the latest edition of $SYSTEMLOG for that date. The program automatically scans $SYSTEMLOG editions backward until it finds an edition that contains the start

date. If none exits, the program aborts with an error message. When it finds the correct edition, it reads forward (across edition boundaries if necessary) until it reaches the EDATE or the end of the current $SYSTEMLOG.

When the date crosses to the next month, the program automatically prints a monthly summary, copies the STATSFOR file, disposes it to the station, starts rewriting the file from the beginning, and continues processing until EDATE.

NOTE

Currently there is no way to gather statistics from DSATE to EDATE without updating the summary file (STATSFOR).

10.4 EXAMPLES

1. Initial run, generate statistics for December, 1978, to present. 12/01/78 date is on $SYSTEMLOG edition 2.

   STATS(ED=02,SDATE=12/01/78)

   This generates a file called 'STATSFOR',ID='SYSTEM'.

2. Last date in 'STATSFOR' is 01/18/79. Gather statistics from that date until 01/20/79)

   STATS(EDATE=01/20/79)

   Lower editions of the $SYSTEMLOG are not scanned, however, if an ED parameter is given. If the start data is not found, the program aborts with an error message.

11.1 INTRODUCTION

JCSDEF is an operating system utility that creates a job class structure from card image directives and/or places a structure into effect. The structure created can be either invoked immediately, saved for later invocation, or both.

Many job class structures can be saved on disk, but only one can be in effect at a time. Each job class structure has a name, a default LIMIT value, and job classes. An installation parameter determines the maximum number of classes that any structure may have. The name of the job class structure appears as a heading on any job class structure display or output. The operator can use the default LIMIT value to define the maximum number of active JXTs via the LIMIT operator command.

11.2 JCSDEF CONTROL STATEMENT

The control statement that calls JCSDEF into execution is:

```
JCSDEF,DIR=dir,DEF=def,LIST=list,INV.
```

Parameters:

DIR=*dir* 1-7 character name of local dataset containing JCSDEF directives; the default is $IN. If DIR=0, no directives are read.

DEF=*def* When DIR≠0, 1-7 character name of local dataset to contain the job class structure definition; when DIR=0, 1-7 character name of local dataset containing the job class structure definition. The default is $JCS.

LIST=*list* 1-7 character name of local dataset to contain all listable output; the default is $OUT. LIST=0 suppresses all listable output.

INV The job class structure is invoked if this parameter is enabled; the default is no invoking of the structure.

The listable output includes diagnostics, informative messages, a list of the directives, and a structure definition.

11.3 JCSDEF DIRECTIVES

JCSDEF recognizes the following directives:

- CLASS Defines a job class.
- SLIMIT Defines a default value for the LIMIT operator command.
- SNAME Assigns a name consisting of 1-7 alphanumeric characters to the structure being defined.

The directives may be in any order. The syntax rules for these directives are the same as those for COS control statements (see CRAY-OS Version 1 Reference Manual).

11.3.1 CLASS DIRECTIVE

The CLASS directive defines a job class. An installation parameter determines the maximum number of job CLASS directives that may be used in a structure definition.

Format:

```
CLASS,NAME=cn,RANK=cr,CHAR=cc,RES=re,MAX=ma,OFF,P=p.
```

Parameters:

NAME=*cn* Job class name consisting of 1-7 alphanumeric characters. The name ALL cannot be used as a class name because it is required in the operator command to specify that all classes within a structure are to be set ON or OFF (for instance, CLASS,ALL,ON). This is a required parameter.

RANK=*cr* Integer rank of this job class; it must be a positive, nonzero number. The highest rank is 1. Each class in a structure must have a unique rank. This is a required parameter.

CHAR=*cc* Expression, including optional relational operators, identifying the job characteristics required by the job class. This is a required parameter. An installation parameter determines the maximum size of an expression. Table 11-1 lists available job characteristics (*ch*) and types that can be used in the descriptors:

DESC1 $\left\{.\text{NOT}.\right\}$ *ch*

where *ch* is a type 1 characteristic.

DESC2 $\left\{.\text{NOT}.\right\}$ *ch* $\left\{\begin{matrix}.\text{EQ}.\\.\text{NE}.\\.\text{GT}.\\.\text{LT}.\\.\text{GE}.\\.\text{LE}.\end{matrix}\right\}$ value

where *ch* is either a type 2A or type 2N characteristic.

A type 2A job characteristic has an alphanumeric value. Alphanumeric substitution rules allow for instances of either of the following:

- A unique character string. For example, 'cccc' matches only 'cccc'.
- A character string containing a variable part. One or more hyphens and/or asterisks are used for the variable part. For example:

  'ccc-' matches any string that begins with 'ccc', including the string 'ccc' itself.

  'ccc*' matches 'cccx' but not 'ccc' or 'ccxx'.

A type 2N job characteristic has a numeric value.

DESC3 $\left\{\begin{matrix}\text{DESC1}\\\text{DESC2}\end{matrix}\right\}\left\{\begin{matrix}\text{.AND.}\\\text{.OR.}\\\text{.XOR.}\end{matrix}\right\}\left\{\begin{matrix}\text{DESC1}\\\text{DESC2}\end{matrix}\right\}$

(i.e. ... ,'M.LT.50.AND.IA',...)

One or more descriptors can be used in the JCC expression. Multiple descriptors must be separated by AND, OR, or XOR
(i.e. 'T.GT.10.AND.T.LT.30.OR.M.LE.75.AND..NOT.IA').

Parentheses may be used and at times must be used throughout the Boolean expression. The expression is evaluated from left to right, with all parenthesized portions evaluated first. NOTs are evaluted first, then ANDs, and finally ORs.

RES=*re* Number of Job Execution Tables (JXTs) to be reserved for use by this class; the default is 0.

MAX=*ma* Maximum number of JXTs to be allocated to this class at a time; the default equals the value of RES. MAX must be greater than or equal to RES.

OFF Disables allocation of JXTs to members of the class until the operator sets it ON. A class that is OFF is treated as if RES and MAX were 0. The default is ON.

P=*p* Replaces each member's job statement priority with the specified class priority (0-15). The default is that job statement priorities are unchanged.

Table 11-1. Job characteristics

ch	Job Characteristics	JOB Parameter	Characteristic Type
M	Field length	M	2N
T	Time limit	T	2N
P	Original priority	P	2N
OLM	$OUT size	OLM	2N
JN	Job name (1-7 characters)	JN	2A
SID	Source ID (1 or 2 characters)	None	2A
DID	Destination ID (1 or 2 characters)	None	2A
TID	Terminal ID (1-8 characters)	None	2A
US	User number (1-15 characters)	US	2A
IA	Jobs marked by the system as interactive. A class defined using IA should be included in every structure definition if interactive jobs exist and are to be given special handling.	None	1
OPP	Jobs marked by the system as having their priority raised to 15 by the operator. A class defined using OPP should be included, with a high rank, in every structure definition if the operator is to be able to move a particular job quickly through the system.	None	1
ORPH	Jobs marked by the system as qualifying for membership in no other class. If a structure does not contain an ORPH class, jobs that do not qualify for membership in a class are marked by the system as having a JOB statement error.	None	1
JSE	Jobs marked by the system as having a JOB statement error. A class defined using CHAR=JSE must be included in every structure.	None	1
SYS	Jobs marked by the system as system jobs. A class defined using SYS should be included in every structure definition if system jobs exist and are to be given special handling.	None	1

11.2.2 SLIMIT DIRECTIVE

The SLIMIT directive defines a default value for the LIMIT operator command. After system startup or whenever LIMIT is needed to start normal job processing, the operator can key in:

- LIMIT,*n* Set LIMIT to *n*.
- LIMIT Set LIMIT to the default value defined for the job class structure in effect.

One SLIMIT directive must be used in a structure definition.

Format:

```
SLIMIT,LI=lim
```

Parameters:

LI=*lim* The default value for the LIMIT operator command. *lim* ranges from zero to the maximum number of JXTs. This is a required parameter.

11.3.3 SNAME DIRECTIVE

The SNAME directive assigns a name to the job class structure being defined. Only one SNAME directive can be used in a structure definition. The name of the job class structure appears as a heading on any job class structure display or output.

Format:

```
SNAME,SN=nm
```

Parameters:

SN=*nm* The structure name. *nm* is 1-7 alphanumeric characters. The default is the dataset name specified by the DEF parameter on the JCSDEF control statement.

11.3.4 EXAMPLE

The directives below define the job class structure DAY:

```
SNAME,SN=DAY.

CLASS,NAME=SYSTEM,RANK=1,CHAR=SYS,RES=1,MAX=1,P=15.

CLASS,NAME=JOBSERR,RANK=2,CHAR=JSE,MAX=3,P=15.

CLASS,NAME=OPERATR,RANK=3,CHAR=OPP,RES=1,MAX=1.

CLASS,NAME=EXPRESS,RANK=4,CHAR='IA.OR.T.LT.7.AND.M.LT.100',
RES=10,MAX=30,P=14.

CLASS,NAME=DEFERRD,RANK=5,CHAR='T.GT.60.AND.M.GT.400',RES=1,
MAX=1,OFF.

CLASS,NAME=NORMAL,RANK=6,CHAR=ORPH,RES=5,MAX=30.

SLIMIT,LI=30.
```

DAY has a recommended LIMIT of 30 and contains 6 classes. Jobs marked by the system as system jobs experience priority throughput and, since RES=MAX=1, are processed one at a time.

Class OPERATR is assigned to the job for which the operator has raised the job priority to 15.

Class JOBSERR contains jobs with a JOB statement error.

Class EXPRESS is assigned only to interactive jobs or jobs using little memory or little CPU time. Jobs in this class should move quickly through the system. They are initiated quickly because the rank is high; they are processed quickly because their priority is set to 14.

Class DEFERRD is OFF during the day and turned on by the operator at some evening hour. While OFF, no JXTs are allocated to its members, and no JXTs are reserved. Once turned ON, this class can allocate only one JXT, so there is no more than one large, long job in the system at any one time.

Class NORMAL has as members all jobs that do not fit into any of the other classes.

CL=NORMAL or CL=DEFERRD on the JOB statement of a user job overrides EXPRESS assignment for that job.

READERS COMMENT FORM

CRAY-OS Ver. 1 System Programmer's Manual (Vol. 2) 2240012 E

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

Cray Research, Inc.
Corporate Addresses

CORPORATE HEADQUARTERS

1440 Northland Drive
Mendota Heights, MN 55120
Tel: 612-452-6650
TLX 298444

≈

THE CHIPPEWA FACILITIES

Manufacturing:
Highway 178 North
Chippewa Falls, WI 54729
Tel: 715-723-2221
TWX 910285 1699

Engineering:
Highway 178 North
Chippewa Falls, WI 54729
Tel: 715-723-5501

CRAY LABORATORIES

Cray Labs Headquarters
5311 Western Avenue
Boulder, CO 80301
Tel: 303-449-3351

Hallie Lab
P.O. Box 169
Chippewa Falls, WI 54729
Tel: 715-723-0266

SALES OFFICES

Eastern Regional Sales Office
10750 Columbia Pike, Suite 602
Silver Spring, MD 20901
Tel: 301-681-9626

≈

Central Regional Sales Office
5330 Manhattan Circle, Suite F
Boulder, CO 80303
Tel: 303-499-3055

Houston Sales Office
3121 Buffalo Speedway, Suite 400
Houston, TX 77098
Tel: 713-877-8053

Austin Sales Office
3415 Greystone, Suite 201
Austin, TX 78731
Tel: 512-345-7034

≈

Western Regional Sales Office
Sunset Office Plaza
1874 Holmes Street
Livermore, CA 94550
Tel: 415-447-0201

Los Angeles Sales Office
101 Continental Boulevard, Suite 456
El Segundo, CA 90245
Tel: 213-640-2351

Seattle Sales Office
536A Medical Dental Building
2728 Colby Avenue
Everett, WA 98201
Tel: 206-259-5075

INTERNATIONAL (SUBSIDIARIES)

Cray Research (UK) Limited
James Glaisher House
Grenville Place
Bracknell, UK
Tel: 44-344-21515
TLX: 848841

Cray Research GmbH
Wartburgplatz 7
8000 Munich 40
West Germany
Tel: 49-89-3630-76
TLX: 05213211

Cray Research Japan, Limited
Shin Aoyama Building, West 1661
1-1 Minami-Aoyama 1-chome
Minato-ku, Tokyo 107 Japan
Tel: 81(03)403-3471

CRAY RESEARCH, INC.