

**GA 18/30
TIME-SHARING
EXECUTIVE
SYSTEM**

GENERAL AUTOMATION, INC.



PRICE \$10.00
88A10059A-A

GA 18/30
TIME-SHARING EXECUTIVE
SYSTEM

GENERAL AUTOMATION, INC.
Automation Products Division

1055 East Street, Anaheim, California 92805 (714) 778-4800

REVISION

<u>Symbol</u>	<u>Description</u>	<u>Approved</u>	<u>Date</u>
A	Programming Release	<i>RCM</i>	July 70

PREFACE

This manual provides an introduction to the Time-Sharing Executive System. It is not a reference manual and, therefore, does not contain extensive detailed information concerning programming for the system. It does provide an overview of the system, describe the interrupt scheme, and explain system generation. The document is organized in two parts. Part I contains the overview of TSS, introduces each major component of the system, and discusses the processing functions. Part II discusses programming techniques for handling various common situations, operating considerations, and system generation procedures. The appendices provide information about certain hardware features and software considerations; system error messages are summarized in one appendix. The document also includes a glossary as well as a bibliography, which lists documents to which the reader may refer for details concerning the implementation and use of the Time-Sharing Executive System.

CONTENTS

PART I - OVERVIEW

Section 1	INTRODUCTION	1-1
1.1	Basic Elements in Process Control	1-2
1.2	Interrupt Hardware	1-3
1.3	Software Priority Concept	1-4
1.4	Servicing an Interrupt	1-5
1.5	Summary	1-6
Section 2	ON-LINE OPERATION	2-1
2.1	Concept of On-Line Operation	2-1
2.1.1	Skeleton Executive	2-1
2.1.2	System Dynamics	2-3
2.2	System Components	2-4
2.2.1	Skeleton I/O	2-5
2.2.2	In-Skeleton Common	2-5
2.2.3	System Director	2-5
2.2.4	User-Written Subroutines	2-6
2.3	User Contributions in Tailoring the Skeleton	2-6
2.4	System Director Components	2-7
2.4.1	Master Interrupt Control Program	2-9
2.4.1.1	Interrupt Service Routine In Skeleton	2-11
2.4.1.2	Interrupt Core Load	2-12
2.4.1.3	Interrupt Service Routine With Core Load	2-12
2.4.1.4	Mainline Core Load	2-12
2.4.2	Program Sequence Control Program	2-12
2.4.3	Interval Timer Control Program	2-14
2.4.4	Time-Sharing Control Program	2-14
2.4.4.1	Selectable Method	2-15
2.4.4.2	Automatic Method	2-15
2.4.5	Error Alert Control Program	2-16
2.5	User-Supplied Subroutines	2-16
2.6	Reentrant Coding	2-17
Section 3	NONPROCESS MONITOR	3-1
3.1	Nonprocess Monitor Operations	3-1
3.2	Components of the Nonprocess Monitor	3-4

CONTENTS (Cont.)

3.2.1	Nonprocess Supervisor	3-4
3.2.2	Disk Utility Program	3-7
3.2.3	FORTRAN Compiler	3-7
3.2.4	Assembler	3-12
Section 4	SYSTEM EVOLVEMENT	4-1
4.1	Temporary Assembled Skeleton	4-1
4.2	System Generation TASK	4-1
4.3	System Generation Overview	4-2
4.4	TASK Disk Write Addresses Program	4-4
4.5	System Loader	4-4
4.5.1	System Loader Monitor	4-4
4.5.2	Table Builder	4-5
4.5.3	Disk Loader	4-5
4.5.4	Disk Edit Phase	4-5
4.5.5	System Loader Error Program	4-5
4.5.6	System Loader Control Cards	4-5
4.5.6.1	Disk Edit Control Card	4-6
4.5.6.2	Assignment Cards	4-6
4.5.6.3	Comment Cards	4-11
4.6	Core Load Builder	4-12
4.7	Skeleton Builder	4-12
4.8	Disk Location Equivalence Tables	4-12
4.9	Cold Start Routine	4-14
4.10	TASK Equate Cards	4-15
4.11	System Director Equate Cards	4-16

PART II - PROCEDURES

Section 5	PROGRAMMING CONSIDERATIONS	5-1
5.1	FORTRAN Subprograms	5-1
5.1.1	SUBROUTINE Subprograms	5-1
5.1.2	FUNCTION Subprograms	5-3
5.2	Assembler Language Subroutines	5-4
5.3	Reentrant Coding	5-6
5.4	Timer Servicing Subroutines	5-7
5.5	Core Loads	5-9
5.5.1	Core Load Coding	5-10
5.5.2	Nonprocess Core Loads	5-10
5.5.3	Process Core Loads	5-17
5.5.4	Interrupt Core Loads	5-18
5.6	INSKEL Interrupt Servicing Routine	5-19

CONTENTS (Cont.)

5.7	Core Load Building	5-19
5.8	Disk Storage Areas	5-22
5.8.1	Core Load Area	5-22
5.8.2	Fixed Location Equivalence Table	5-22
5.8.3	Working Storage	5-24
5.8.4	FORTRAN Disk Input/Output	5-24
5.8.5	Assembler Disk Input/Output	5-29
5.9	Subroutine Library	5-32
5.9.1	Card I/O Subroutine - CARDN	5-34
5.9.2	Disk Storage Subroutine - DISKN	5-35
5.9.3	Printer/Keyboard Subroutine - TYPEN, WRTYN	5-36
5.9.4	Printer Subroutine - PRNTN	5-38
5.9.5	Magnetic Tape Subroutine - MAGT	5-39
5.9.6	Paper Tape I/O Subroutine - PAPTN	5-40
5.9.7	Plotter Subroutine - PLOTX	5-41
5.10	Summary of DUP Operation	5-41
5.10.1	DEFINE Routine	5-41
5.10.1.1	Object Core Size	5-42
5.10.1.2	Number of Disk Drives	5-42
5.10.1.3	Disk Area Configuration	5-43
5.10.1.4	Remove a Processor	5-45
5.10.1.5	Condense Relocatable Program Area	5-45
5.10.2	DLABL Routine	5-45
5.10.3	STORE Routine	5-46
5.10.4	STOREDATA Routine	5-47
5.10.5	STORECI Routine	5-48
5.10.6	STOREMD Routine	5-49
5.10.7	DUMP Routine	5-50
5.10.8	DUMPDATA Routine	5-51
5.10.9	DUMPLET Routine	5-51
5.10.10	DELET Routine	5-51
5.10.11	SEQCH Routine	5-52
5.10.12	DICLE Routine	5-53
5.10.13	DWRAD Routine	5-53
5.11	Common Areas	5-54
Section 6	OPERATING CONSIDERATIONS	6-1
6.1	Operating TASK Off-Line	6-1
6.2	TASK Disk Write Addresses Routine	6-2
6.3	TASK Disk Duplication Program	6-5
6.4	System Cold Start	6-6
6.4.1	Cold Start Name Card	6-6
6.4.2	Cold Start Procedure	6-7
6.5	Clearing Core	6-8

CONTENTS (Cont.)

Section 7	SYSTEM GENERATION	7-1
7.1	Summary of Generation Procedures	7-1
7.2	System Generation Components	7-1
7.2.1	Supplied TSS System	7-1
7.2.2	User-Prepared Control Cards	7-3
7.3	System Generation Procedures	7-3
7.3.1	Loading TASK and Writing Disk Addresses	7-3
7.3.2	Loading the Supplied Decks on Disk	7-11
7.3.3	Assembling TASK	7-11
7.3.4	Assembling the System Director	7-11
7.3.5	Defining the System Configuration	7-26
7.3.6	Compiling Skeleton Subroutines	7-26
7.3.7	Building the Skeleton	7-26
7.3.8	Compiling Process Programs	7-30
7.3.9	Building Process Core Loads	7-30
7.3.10	On-Line Cold Start	7-30
7.3.11	Storing Relocatable Programs on Disk from Cards	7-30
7.3.12	Building a Nonprocess Monitor Disk Pack	7-47
7.3.13	Off-Line Cold Start	7-47
Appendix A	System Error Messages	A-1
Appendix B	Calling Sequences for System Routines	B-1
Appendix C	Differences Between TSS and IBM's TSX	C-1
Glossary		G-1
Bibliography		

ILLUSTRATIONS

1-1	Multilevel Interrupts	1-4
2-1	Core Map, Illustrating Skeleton Executive	2-2
2-2	Typical Skeleton Executive	2-4
2-3	System Director Components	2-8
3-1	Nonprocess Monitor Storage (On-Line System)	3-2
3-2	Nonprocess Monitor Storage (Off-Line System)	3-3
3-3	Nonprocess Monitor Components	3-4
5-1	Build and Execute a Type 1 Nonprocess Core Load	5-14
5-2	Build and Execute a Type 2 Nonprocess Core Load	5-15
5-3	Delete a Type 2 Nonprocess Core Load	5-16
5-4	Disk Storage Arrangement	5-23

ILLUSTRATIONS (Cont.)

7-1	System Generation Flowchart	7-2
7-2	Supplied System Object Decks	7-4
7-3	ZAP Card	7-8
7-4	TASK High Core Loader Cards	7-9
7-5	Sequence of Control Cards and System Decks for TSS System Load	7-19
7-6	TASK Source Deck and Equate Cards	7-22
7-7	System Director Source Deck and Equate Cards	7-25
7-8	Skeleton Builder Object Deck and Control Cards	7-36
7-9	On-Line Cold Start	7-43
7-10	Cold Start Cards	7-44
7-11	Off-Line Cold Start	7-51

TABLES

2-1	Priority Interrupt Level Structure and Assignment	2-9
3-1	Nonprocess Supervisor Control Records	3-5
3-2	Disk Utility Program Control Statements	3-8
3-3	DUP Routines Control Records	3-10
3-4	FORTTRAN Control Records	3-13
3-5	Assembler Control Records	3-16
4-1	Interrupt Assignment Code/Logical Unit Number Assignments	4-8
4-2	Group 1 TASK EQU Cards	4-18
4-3	Group 2 TASK EQU Cards	4-25
4-4	Error Alert Control Printer Combinations	4-26
4-5	System Director Equate Cards	4-27
5-1	Summary of Capabilities and Restrictions of Nonprocess Core Loads	5-12
7-1	Loading TASK in Core	7-5
7-2	Writing Disk Addresses	7-12
7-3	Loading the Supplied System Decks	7-16
7-4	Assembling TASK	7-20
7-5	Assembling the System Director	7-23
7-6	Defining the System Configuration	7-27
7-7	Compiling Skeleton Subroutines	7-29
7-8	Building the Skeleton	7-31
7-9	Compiling Process Programs	7-37
7-10	Building Process Core Loads	7-38
7-11	On-Line Cold Start	7-39
7-12	Storing Relocatable Programs on Disk from Cards	7-45
7-13	Building a Nonprocess Monitor Disk Pack	7-48
7-14	Off-Line Cold Start	7-49

SECTION 1 - INTRODUCTION

The Time-Sharing Executive System (TSS) is a FORTRAN-oriented disk-resident operating system that enables the user to make optimum use of a GA 18/30 Industrial Supervisory System in controlling processes and complex environments. In addition to this on-line capability, the time-sharing aspect of the TSS allows the execution of low priority jobs (such as assembling or compiling programs) under the control of a batch-processing monitor. Such operation is referred to as "background" operation as contrasted with "foreground" operation, which is the primary function of TSS: process control.

Another capability included in the Time-Sharing Executive System is off-line operation. TSS provides the user with a monitor that enables him to operate his GA 18/30 Computer apart from the process it may normally control. Thus, the equipment is available for any computing function whenever the process is not running.

In industrial control systems individual installation requirements vary from one installation to another. These differences may be in the hardware configuration or in dissimilarities inherent in the application. Therefore, each installation must be defined, or tailored, for its specific function requirements and input/output configuration. The modular design of TSS enables the user to include or exclude any functions he wishes. Furthermore, user-written programs can be easily incorporated in the system. The final result of the tailoring function is an efficient operating system, unique to the installation.

1.1 BASIC ELEMENTS IN PROCESS CONTROL

Basically, all on-line real-time control systems behave in much the same fashion. The computer reacts to input data from a real world environment and outputs data to correct or control that environment. Emergency conditions are also sensed and appropriate action is initiated. Status sensing, data computation, and reaction control must occur within a specified interval of time to prevent disruption of the process. Generally, the system's capability is determined by how well it is able to respond.

All input and output operations of the GA 18/30 Computer (including data transfer, interrupt control, and certain internal control operations) are initiated by one multipurpose, input/output instruction: Execute I/O (XIO). Thus, all communication between the real world environment and the computer is through the XIO instruction. The programmer who writes the process control programs and the interrupt servicing routines uses the XIO instruction to perform a variety of functions:

- Sense the operational status of an I/O device, process, or internal condition, or sense devices requesting interrupt recognition
- Control (change) the operating condition of an I/O device or internal feature
- Read data from an input device into memory
- Output data from memory to a device

The programmer works with the hardware interrupt feature to accomplish his purposes.

1.2 INTERRUPT HARDWARE

TSS provides a multi-interrupt priority control scheme, consisting of a hardware priority structure, data storage areas in core for each interrupt level, and a program to recognize, control, and direct the servicing of interrupts. The hardware priority structure provides for 2 internal and up to 24 external interrupt levels, which the user may assign to I/O, process, or programmed interrupts. Up to 16 interrupt request lines can be connected to each interrupt level (except trace). Each interrupt level (except trace) may have an interrupt level status word (ILSW) of up to 16 bits to identify the source of the interrupt request. Each I/O device or process has a 16-bit word to identify -- among other conditions -- the specific condition responsible for an interrupt request. The status word for an I/O device is called a device status word (DSW); that for a process is called a process interrupt status word (PISW).

Thus, each interrupt request line is positioned by order of priority (as defined in paragraph 2.4.1 and table 2-1). The highest priority is closest to the output, and the lowest priority is furthest away. When an interrupt request is received at a given level and if no higher priority level is presently being served, the control scheme permits the interrupt request line to be activated. A unique address associated with that particular level is supplied to the system, which transfers control to that location. The return address for the interrupted program is preserved, and then the Master Interrupt Control Program (see paragraph 2.4.1) is executed to direct the servicing of the interrupt. After the interrupt has been serviced, control is returned to the interrupted program (see figure 1-1).

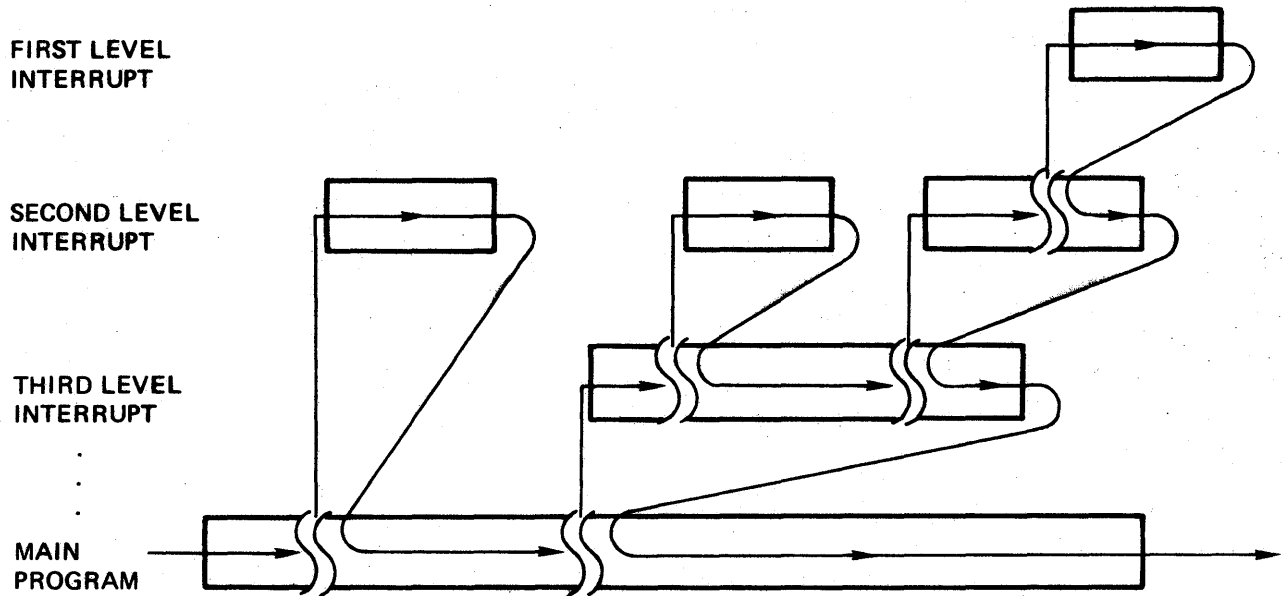


Figure 1-1. Multilevel Interrupts

The user has the facility to mask interrupt levels. Masking inhibits interrupts to the computer. The user can inhibit or permit specified levels of interrupts and can determine the status of interrupt levels (i. e., inhibited or not) at any time. Although a level may be masked, the fact that an interrupt has occurred is not lost. Thus, when a level is unmasked, a pending interrupt can be received.

1.3 SOFTWARE PRIORITY CONCEPT

The user writes the process programs for his installation. Additionally, he provides routines to service interrupts which can occur. The way interrupts are assigned to the interrupt levels largely determines the method of programming used to service the interrupts. Essentially, there are four types of service routines, determined by method of access. An interrupt service routine may reside permanently in an area of core reserved for it by the Time-Sharing Executive System; it may reside on disk and be loaded into core when its interrupt occurs; it may be stored as a subroutine with a main program and be loaded from disk each time the main program is loaded; or a main program may

service an interrupt that was recorded when it occurred but not serviced immediately. (These types of service routines are discussed in more detail in paragraph 2.4.1.) The first type of subroutine is the fastest (in response time) and should be used for the most critical interrupts. The second type is slower, since the interrupted program must be stored on disk and the subroutine must be loaded into core before it can be executed. The third type is as fast as the first type if the main program is in core when the interrupt occurs; otherwise, it is as slow as the second type. The fourth type of subroutine is slowest and is used for the most infrequent and least critical interrupts.

1.4 SERVICING AN INTERRUPT

When an interrupt is detected at the hardware level, a portion of TSS, the Master Interrupt Control (MIC) Program, assumes control for the servicing of that interrupt. The MIC program:

- Saves the interrupted registers when an interrupt is processed on the appropriate work level
- Directs the interrupt to its servicing routine
- Restores the FORTRAN I/O buffers (if required)
- Restores the interrupted registers
- Returns control to the interrupted program

For example, assume an interrupt was assigned by the user to level five. The following events occur when that interrupt is recognized.

1. The GA 18/30 Computer recognizes interrupt requests at the completion of the current instruction cycle. At that time an indirect branch (BSI) through a fixed location in core is executed. This location contains the starting address of the level work area associated with level five. The instructions in this work area set the level busy, save Index Registers 1, 2, and 3, and set in Index Register 3 a pointer to this work level. It is through the level work area that an interrupt formally enters MIC.

2. After the various registers of the interrupted program have been saved, the problem remains of determining which of 16 possible interrupts is to be serviced on this level. This determination is made by sensing the ILSW.
3. MIC reads the ILSW for level five and determines which is the left-most nonzero bit in the ILSW. Then MIC determines where the interrupt service routine associated with that nonzero bit is stored (i.e., on disk or in core -- and where in core).
4. MIC transfers control to the appropriate interrupt service routine (after loading it from disk, if necessary).
5. After performing its function, the subroutine returns control to MIC via a special statement (CALL INTEX, see section 5) which includes a branch or skip on condition instruction (BOSC) with bit 9 set to 1, permitting lower interrupt levels to interrupt the computer.

During the entire time -- from the occurrence of the interrupt to the execution of the BOSC -- the computer is said to be operating at priority level five and cannot be interrupted by any lower priority interrupts.

6. MIC reloads the interrupted program, if necessary; i.e., if the interrupt service routine had to be loaded from disk for execution, MIC would first store the interrupted program in a special save area on disk so it would not be destroyed when the service routine is read into core. Finally, control is returned to the program at the instruction following the one where the interrupt occurred.

1.5 SUMMARY

Real-time computation has been defined as a situation in which input data change with time so that the execution of the program affects the answers derived by the program. Job-dependent programs are not real-time; time-dependent and interrupt-dependent programs are real-time. Time-dependent programs must make decisions based on the time of day, and interrupt-dependent programs must

respond to interrupts originating at unpredictable times in the world outside the computer. The Time-Sharing Executive System is a set of programs designed to provide programming flexibility in a real-time environment. TSS relieves the user of much of the required system programming effort, freeing him to concentrate on the primary task of problem solution. Thus, TSS is the interface between the hardware and the controlled process for on-line operation and between the user and the operating system for off-line operation.

SECTION 2 - ON-LINE OPERATION

In a real-time environment the processor controller receives inputs randomly from the monitored process. In response to those inputs, the computer returns an output to the process. Thus, the concept of real-time implies that a processor controller responds to inputs as they occur in the "outside world."

2.1 CONCEPT OF ON-LINE OPERATION

The Time-Sharing Executive System (TSS) operates in an on-line mode under control of the Skeleton Executive. The Skeleton Executive is the basis, or framework, of an on-line TSS system. It must be resident in core storage; i. e., the Skeleton Executive must be in core storage before real-time processing can take place. The Skeleton Executive accepts input, determines which portion of the system is needed to process that input, and brings that portion from disk storage into core to perform the required function.

2.1.1 Skeleton Executive

The structure of the Skeleton Executive is flexible and is determined by the user at system generation time. "System generation" is the process of assembling some number of routines to form the system. In generating the Skeleton Executive, the user has numerous options; e. g., he may include frequently used subroutines, rapid response interrupt servicing routines, or other user-written routines. System generation is discussed in more detail in section 4, and the procedures for performing this operation are given in section 7. Figure 2-1 illustrates the area of core storage that the Skeleton Executive occupies.

The content of the Skeleton Executive depends upon the application for which it is to be used, because it contains both supplied and user-written routines. The supplied routines are considered to be the operating system. The user-written routines perform the actual control of the processes. Thus, the supplied routines

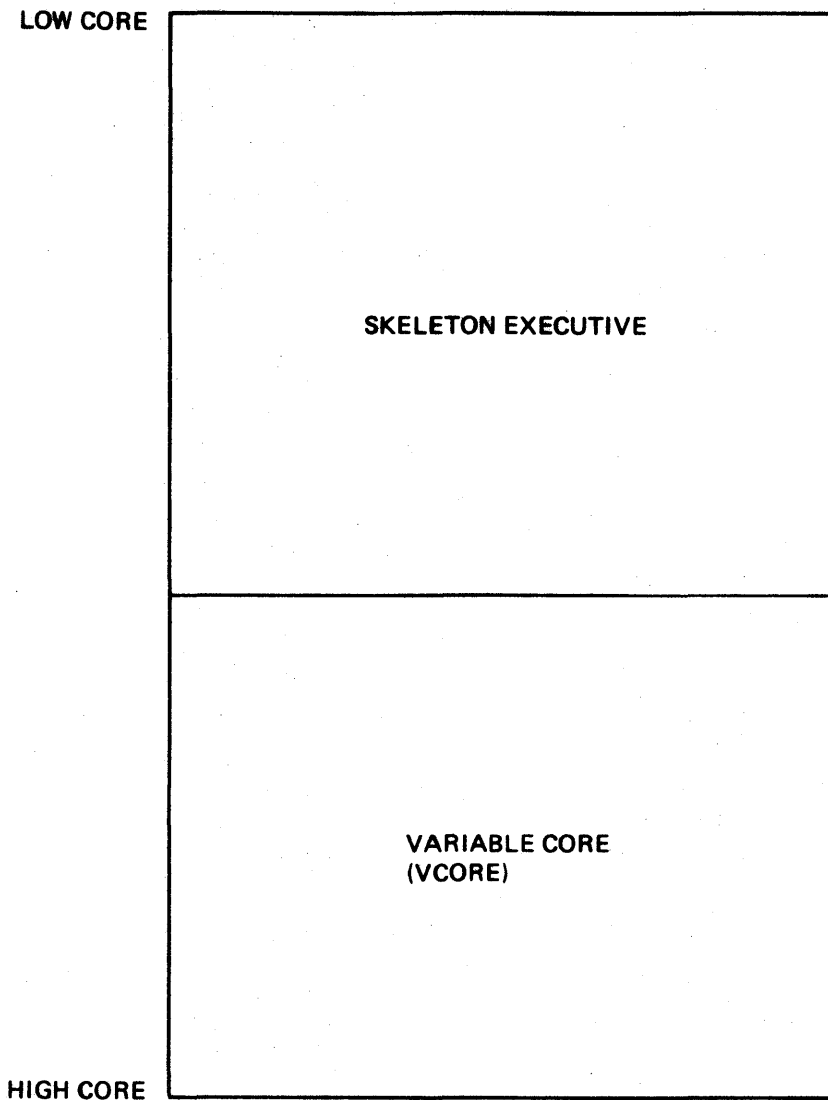


Figure 2-1. Core Map, Illustrating Skeleton Executive

(i. e. , operating system) coordinate the execution of the user-written routines and act as an interface between the actual hardware (interrupt structure, timers, etc.) and the user-written routines.

That portion of core memory not occupied by the Skeleton Executive is called variable core or VCORE. The Skeleton Executive loads user routines from disk storage into VCORE when those routines are to be executed. The size of VCORE is not intended to be sufficiently large to store at one time all the instructions required for the execution of all functions. Instead, the routines

must be segmented into units, called core loads, which are stored on disk in executable core image format. Thus, a core load may be read from disk and executed, and it, in turn, may request other core loads be brought into core and executed to complete the function. There are four types of core loads commonly used in TSS :

- Mainline core load
- Interrupt core load
- Combination core load
- Nonprocess core load

A mainline core load is one that does not directly service an interrupt (e.g., analysis programs, logging programs, etc.); it executes on the lowest interrupt level. An interrupt core load is a program unit that resides on disk and is brought into core to service a particular interrupt. A combination core load is one that can be executed as either an interrupt or a mainline core load. A nonprocess core load is one that is controlled by the Nonprocess Monitor, discussed in section 3. (The procedure for creating core loads, storing them on disk, and establishing system communication for them is discussed in part II.)

2.1.2 System Dynamics

After the user has generated his TSS system and stored it on disk, he must initiate or start system operation. This process is referred to as "cold start." Cold start procedures are described in detail in part II. A cold start routine is supplied that enables the user to load the Skeleton Executive from disk to core. Then a user-written core load is brought into VCORE to perform initialization procedures (such as setting timers and indicators, etc.). After the cold start is accomplished, the system operates without operator intervention under control of the Skeleton Executive. Interrupts from the process are handled by the Skeleton Executive routines which may initiate core swapping (i.e., the storing of the contents of VCORE onto disk and transferring of a core load from disk to VCORE.). If any hardware error occurs -- during input from the process or during core swapping -- the Skeleton Executive automatically takes appropriate action to respond to that error.

2.2 SYSTEM COMPONENTS

As noted before, the content of the Skeleton Executive depends upon the application the system is to control. The content and, thus, the size of the Executive are fixed at system generation time. A typical Skeleton Executive would include the parts illustrated in figure 2-2.

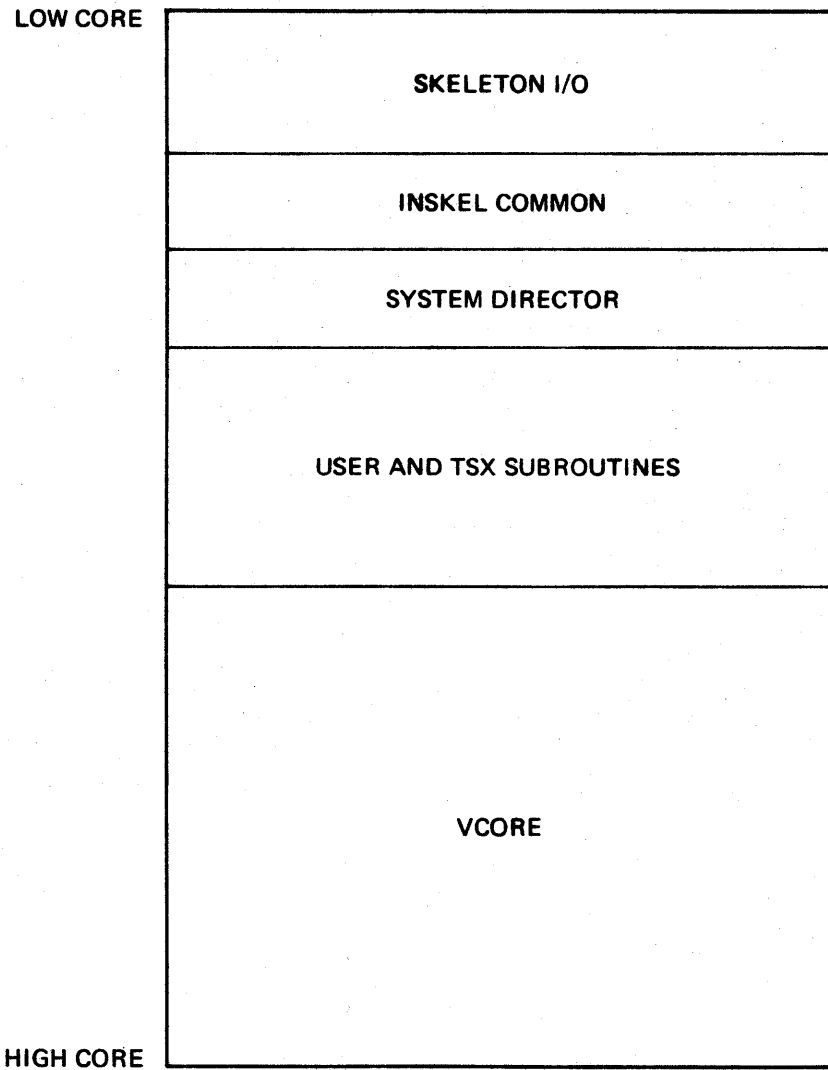


Figure 2-2. Typical Skeleton Executive

2.2.1 Skeleton I/O

The Skeleton I/O portion is a set of input/output routines that provide communication between the CPU and the various data processing peripherals (e. g. , card reader, card punch, disk, printer) for the transfer of data. These routines are used by both supplied and user-written routines. A basic set of routines is supplied with the system and normally constitutes the skeleton I/O; however, the user can include other routines at system generation time. The basic set of supplied routines is:

- Disk Storage Subroutine (DISKN) – performs all reading from and writing to disk storage
- Printer/Keyboard Subroutine (TYPEN/WRTYN) – transfers data to and from teletypewriters
- Printer Subroutine (PRNTN) – controls all print and carriage positioning functions for the line printer
- Card Subroutine (CARDN) – controls input from the punched card reader

These four subroutines are provided with and used by the system.

2.2.2 In-Skeleton Common

The In-Skeleton Common (INSKEL) provides communication among the various user-written core loads. In assigning variables to this area, the user must use the unique label INSKEL in a FORTRAN COMMON statement. This area can be referenced by any process or nonprocess program under the on-line system.

2.2.3 System Director

The System Director is the basic component of the Skeleton Executive and controls all facets of process monitoring. The System Director resides in core

at all times, and all permanent areas are storage protected to avoid being accidentally destroyed. The System Director is the primary control unit of TSS and, as such, it:

- Directs the servicing of interrupts
- Supervises the execution of core loads
- Monitors the interval timers
- Processes errors

Further, when there is no process control to be monitored, the System Director makes VCORE available for execution of background jobs.

2.2.4 User-Written Subroutines

The user may include in the skeleton area any supplied or user-written subroutines he chooses. These may be interrupt subroutines, timer subroutines, trace and error subroutines, arithmetic and conversion subroutines, etc. The decision to include a routine in this area instead of storing it on disk as part of a core load is influenced by such factors as:

- Size of subroutines (the larger the skeleton area, the smaller the VCORE area for executing core loads)
- Required response time (subroutines in the skeleton do not require time for loading from disk as core loads do)
- Frequency of use (core storage of frequently used subroutines avoids excessive core swapping)

User-written subroutines that are to be included in the Skeleton Executive must be compiled or assembled in relocatable format and placed on disk prior to system generation.

2.3 USER CONTRIBUTIONS IN TAILORING THE SKELETON

Because the user defines the TSS system in terms of his own application, each installation is unique. Differences between installations may take the form of

- Different applications
- Different core storage sizes
- Different peripheral equipment
- Different priorities
- Different throughput requirements

Therefore, each installation must be tailored for the specific hardware configuration and process requirements. It is the user's responsibility to define the Skeleton Executive for his installation via a process called system generation. Before system generation time, however, the user must determine what features he wishes to include in his TSS system, because these considerations directly affect the size and composition of the system.

Basically, three things determine the size of the skeleton:

- Hardware configuration
- Number and size of user-written subroutines
- Amount of INSKE L common

The hardware configuration must be described to the system generator in terms of core size and available peripherals. In determining what user-written subroutines are to be included in the skeleton, the user would consider such questions as: which interrupts require the fastest response time and, therefore, should be handled by in-skeleton subroutines; and, which subroutines should be in core -- because of frequency of use -- to avoid excessive core swapping. The amount of INSKE L common storage depends upon both user-written and supplied routines' requirements. The methods used to specify all required information at system generation time are explained in section 7.

2.4 SYSTEM DIRECTOR COMPONENTS

The System Director is a group of supplied programs that constitute the nucleus of the TSS system. The user must understand the functions of the components of the System Director to utilize fully the TSS system. The five component programs (figure 2-3) are:

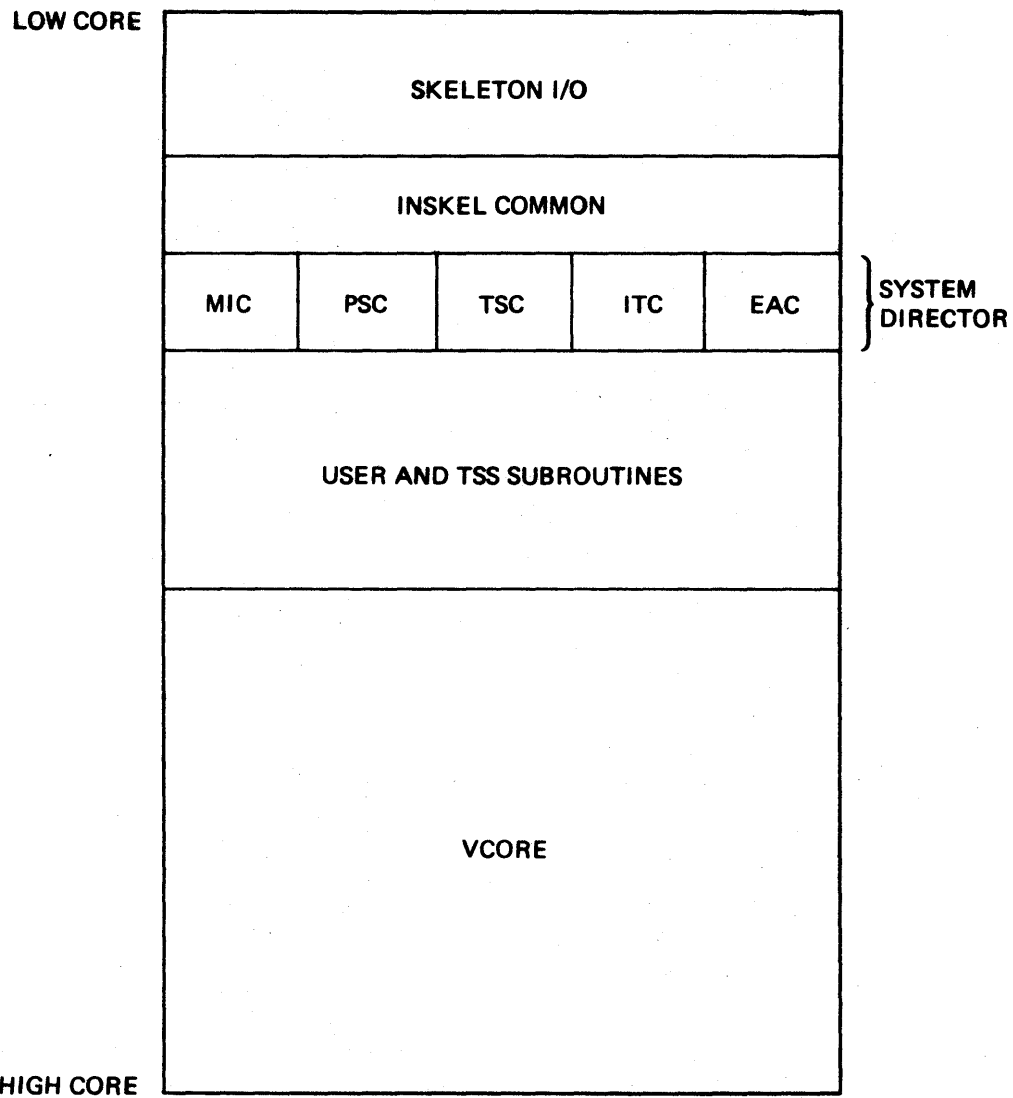


Figure 2-3. System Director Components

- Master Interrupt Control (MIC)
- Program Sequence Control (PSC)
- Time-Sharing Control (TSC)
- Interval Timer Control (ITC)
- Error Alert Control (EAC)

These programs are discussed in the following paragraphs from a functional viewpoint; details concerning programming considerations are provided in section 5.

2.4.1 Master Interrupt Control Program

The Master Interrupt Control (MIC) program is the basic interface between the hardware interrupts and the user-written programs or the system interrupt handlers. In the TSS system there are three essential elements to providing multi-interrupt priority control:

- A hardware priority scheme
- A data storage area in core for each interrupt level
- The MIC program to service the interrupts

The hardware priority structure provides for 2 fixed and up to 24 additional interrupt levels, which the user may assign to I/O or programmed interrupts, as shown in table 2-1.

For each priority level used, the system has (in the skeleton) a level work area. In this area MIC saves the status of the interrupted program. Further, MIC sets up the status of the level work area for each level so that a system subroutine, using the level work area for intermediate storage, can be entered (executed) from all priority levels. (Paragraph 2.6 discusses reentrant coding.)

From a programming standpoint there are three classes of interrupts: internal, input/output, and external (or process). Internal interrupts are those associated with errors within the central processing unit, such as parity error or detection of an illegal operation code. Interrupts of this class are serviced by supplied

Table 2-1. Priority Interrupt Level Structure and Assignment

Interrupt†	Priority Level††	Decimal Address	Program Interrupt	I/O, Timer, Process Interrupt: Assignment Allowed	
Internal Trace	1	8	No	No	
	26	9	No	No	
	Assigned Levels	0	11	Yes	Yes
		1	12	Yes	Yes
		2	13	Yes	Yes
		3	14	Yes	Yes
4		15	Yes	Yes	
5	16	Yes	Yes		
BASIC	6	17	Yes	Yes	
	7	18	Yes	Yes	
	8	19	Yes	Yes	
	9	20	Yes	Yes	
	10	21	Yes	Yes	
	11	22	Yes	Yes	
	12	23	Yes	Yes	
	13	24	Yes	Yes	
	14	25	Yes	Yes	
	15	26	Yes	Yes	
	16	27	Yes	Yes	
	17	28	Yes	Yes	
	18	29	Yes	Yes	
	19	30	Yes	Yes	
	20	31	Yes	Yes	
	21	32	Yes	Yes	
	22	33	Yes	Yes	
	23	34	Yes	Yes	
OPTIONAL	6	17	Yes	Yes	
	7	18	Yes	Yes	
	8	19	Yes	Yes	
	9	20	Yes	Yes	
	10	21	Yes	Yes	
	11	22	Yes	Yes	
	12	23	Yes	Yes	
	13	24	Yes	Yes	
	14	25	Yes	Yes	
	15	26	Yes	Yes	
	16	27	Yes	Yes	
	17	28	Yes	Yes	
	18	29	Yes	Yes	
	19	30	Yes	Yes	
	20	31	Yes	Yes	
	21	32	Yes	Yes	
	22	33	Yes	Yes	
	23	34	Yes	Yes	

†All levels except trace have interrupt level status words, and all levels except internal and trace can be masked.

††Priority levels are numbered from highest (1) to lowest (26).

routines as soon as they are recognized. Input/output interrupts are associated with peripheral devices. External interrupts are those associated with the process and programmed interrupt features. They are serviced by one of four types of user-written routines:

- Interrupt service routine in skeleton
- Interrupt core load
- Interrupt service routine included with core load
- Mainline core load

When the user defines the system, he must designate the way each interrupt is to be handled, to enable MIC to service the interrupts properly.

2.4.1.1 Interrupt Service Routine In Skeleton

During system generation the user can include interrupt servicing subroutines in the skeleton. This group normally comprises the majority of the skeleton routines. This type of interrupt servicing requires less time than the other three types; i.e., less time elapses between the instant the interrupt signal is recognized and the instant an instruction is executed to begin the appropriate reaction.

Interrupt processing occurs in the following sequence. A signal is received from the process. MIC responds to the interrupt signal, determines that the interrupt is to be handled by a user-written routine that is in the skeleton, and transfers control to that routine. The routine performs the necessary action (e.g., prints a message, outputs information, initiates the correction of a process condition, etc.) and then returns control to MIC, which in turn returns control to the interrupted program. MIC stores the contents of the registers that reflect the status of the machine at the time of the interrupt and restores these registers after the user's interrupt routine has performed its function, but before transferring control to the interrupted program.

2.4.1.2 Interrupt Core Load

The user can designate a core load as an interrupt core load. In this case MIC saves the status of the machine and the contents of VCORE in an interrupt save area on disk before loading the user's interrupt core load. The interrupt core load (once in core) responds in the same way an interrupt service routine in the skeleton would. When its function is completed, the interrupt core load returns control to MIC, which restores the machine registers and VCORE to their previous states (i. e. , before the interrupt occurred) and returns control to the interrupted program.

The response time for this method of interrupt servicing includes the time for the core exchange and is, therefore, not as fast as in-skeleton interrupt service routines.

2.4.1.3 Interrupt Service Routine With Core Load

A mainline core load may have included with it subroutines that service interrupts. If the mainline core load is in core when such an interrupt occurs, MIC uses that subroutine just as if it were included in the skeleton. An interrupt core load that can process the same interrupt must also exist. Then, if the mainline core load with the service routine is not in core when the interrupt occurs, MIC can load the interrupt core load (in the manner explained in paragraph 2.4.1.2) to service the interrupt.

2.4.1.4 Mainline Core Load

If neither an interrupt core load nor an in-skeleton servicing routine has been assigned for a process interrupt, MIC records the fact that the interrupt occurred. Such interrupts can be serviced by a mainline core load as explained below under "Program Sequence Control Program."

2.4.2 Program Sequence Control Program

Core loads are user-written modules on disk that are brought into core by the Program Sequence Control (PSC) portion of the System Director. Some core

loads are designed to respond to interrupts. The most common kind of core loads are those designed to execute on the mainline level. (The process by which these modules are prepared and placed on disk is described in section 4. However, for purposes of this discussion, assume that this process is defined by the term "built" so that a core load that has been "built" exists on disk.) When each core load is built, the level on which it is to be executed is defined. Mainline core loads operate on the lowest priority level.

In process control, actual timely response is accomplished by using in-skeleton routines and interrupt core loads. Other functions (reports, file manipulations, analyses, etc.) must also be performed and in an orderly manner; however, their importance does not require one to interrupt the other. A method of sequencing these jobs is provided by PSC. To perform its functions, PSC utilizes a storage area called the Mainline Core Load Queue Table (referred to as "the queue"). This table contains a list -- ordered by name and priority of function -- of the mainline core loads to be executed. Placing entries into and removing them from the queue is not the responsibility of PSC. The function of PSC is to initiate operation of the highest priority job by loading the appropriate mainline core load and executing it when space in VCORE is available.

Subroutines are provided that enter or remove core loads from the mainline queue. These subroutines can be used by interrupt service routines as well as mainline programs. Therefore, a job sequence may be initiated from any level. The characteristics and calling sequences of the subroutines are explained in section 5.

A particular job may require more than one core load for its completion. Essentially, two routines are supplied for the sequencing of jobs that require multiple core loads: CHAIN and SPECL.

Chaining allows the present core load to be overlaid by the next sequential core load. The first core load initiates the chaining process by a call to the CHAIN subroutine.

The SPECL subroutine of PSC provides the second method of sequencing. This subroutine terminates the current core load, saves it in a special save area, stores the core load status, and executes the core load associated with the call to SPECL. The core load that is brought into VCORE by SPECL may return control to the saved core load automatically by a call to the BACK subroutine of PSC. However, the new core load is not required to return control to the saved core load; it may call other core loads or may end the job sequence. Ending the job sequence is accomplished by a call to the VIAQ subroutine of PSC. This call causes PSC to load the highest priority core load listed in the queue and to transfer control to it.

2.4.3 Interval Timer Control Program

The GA 18/30 hardware provides three machine timers, designated A, B, and C. Each timer is assigned a specific time period:

Timer A = 0.1 millisecond

Timer B = 1.0 millisecond

Timer C = 10.0 milliseconds

Timers A and B are available to the user's programs. Timer C is used by the TSS system to provide nine programmed timers and a real-time clock. Therefore, 11 timers are actually available to the user. The Interval Timer Control (ITC) program provides for control of the three hardware timers. ITC also performs other functions; namely, it tests for no response from the Teletype, resets the operations monitor during time-sharing, and performs end of time-sharing (see section 3 for a discussion of time-sharing under TSS).

2.4.4 Time-Sharing Control Program

In most installations there will be a considerable amount of time that is not used for process control. TSS provides a time-sharing feature to enable the user to execute low priority jobs (e.g., assembling or compiling programs) during that "idle" time. The Time-Sharing Control (TSC) program monitors the execution of low priority jobs (i.e., nonprocess jobs) and automatically interrupts them when a higher priority job (i.e., any process control function) must be

executed. Thus, with TSC the user can perform batch processing without taking his system off-line.

When such idle time is available in the system, control can be automatically transferred to the Nonprocess Monitor, an independent system that is similar to any batch (stack-job) monitor system. TSC is the portion of the System Director which, in conjunction with the ITC program, allocates VCORE for batch processing use. There are essentially two ways in which the Nonprocess Monitor can obtain time (and, thus, access to VCORE) for its use:

- Selectable method
- Automatic method

2.4.4.1 Selectable Method

Process programs (mainline core loads only) can initiate time-sharing for a specific period of time by a call to the SHARE subroutine of TSC. This selection of time-sharing is used for special applications where time-sharing is desired without the use of the queueing technique. The time-sharing operation, initiated by a call to SHARE, terminates when the time interval specified by the user has elapsed; however, interrupts are serviced as they occur, and an interrupt routine can terminate time-sharing mode by a call to the ENDTS subroutine. (Calls to these subroutines are discussed in part II.)

2.4.4.2 Automatic Method

VCORE automatically becomes available to the Nonprocess Monitor when the VIAQ subroutine (see part II) of PSC checks the queue and determines that no core loads are queued for execution. In order for the VIAQ subroutine to initiate time-sharing, the user must indicate through the use of console switches that batch jobs are to be handled. For this method the period of time allocated for time-sharing is specified by the user when the System Director is assembled at system generation time.

2.4.5 Error Alert Control Program

The Error Alert Control (EAC) program is the part of the System Director that:

- Receives error interrupts
- Analyzes the type of error (e. g. , an I/O error that persists despite repeated corrective action by an I/O subroutine; an internal machine error, such as invalid operation code or parity; and other control subroutine error conditions, such as FORTRAN I/O)
- Saves the machine status at the time of the error so that, after the error has been corrected, processing can be reinitiated without loss of information
- Determines operating conditions (e. g. , process or nonprocess mode, availability of user-written error subroutine)
- Selects the appropriate recovery procedure (e. g. , continue processing, restart, reload)
- Produces error messages

EAC also has the capability to dump VCORE to disk if this option is elected when the System Director is assembled at system generation time.

(See appendix A for information concerning error messages.)

2.5 USER-SUPPLIED SUBROUTINES

As stated earlier, the user may specify that certain subroutines are to be included in the Skeleton. Other subroutines are assembled or compiled and stored on disk, to be loaded into VCORE along with the core load that uses them. However, if a program requires a large number of subroutines, VCORE may not provide sufficient space for them. To avoid this problem, TSS provides the capability of loading a subroutine from disk into core at the time the executing program calls that subroutine. Such a subroutine is referred to as a LOCAL

(load-on-call) subroutine. All LOCALs called by a core load program are stored in the same core area; i. e., the second LOCAL subroutine overlays the first one, the third overlays the second, etc. The effect, then, is that LOCALs enable the user to have a larger program than would otherwise be possible. (See "Section 5 - Programming Considerations," for examples.)

2.6 REENTRANT CODING

It is possible that core loads that are executed on different levels may call the same subroutine. To allow a subroutine to be entered at any time and on any interrupt level, some method of reentrant coding must be used. All TSS system subroutines are reentrant and can be called repeatedly by different interrupt routines on different levels. Users may write reentrant routines for their core loads.

The method of reentrant coding employed in TSS uses the level work areas. A level work area of 104 locations is provided for each interrupt level specified by the user. A level work area for a given interrupt level can be used only by programs operating on that level. Of the 104 locations that constitute a level work area, the first 62 are reserved for TSS use; the remaining 42 are available for use by other programs. The starting address of the level work area for any interrupt level is always in location 104 (68_{16}). Thus, an index register, loaded with the contents of that location, should be used to reference all temporary storage, i. e., the 42 temporary locations available to users' programs. If a subroutine being executed is interrupted and the interrupt servicing routine calls that subroutine, there will be no storage conflict, because MIC always sets location 104 to the correct level work area address for each interrupt serviced. (Details on ways to safeguard partial results and other considerations in writing reentrant code are presented in section 5.)

SECTION 3 -- NONPROCESS MONITOR

The Nonprocess Monitor is an independent programming system, designed to operate in one of two modes within the TSS system:

- On-line - In the on-line mode the Nonprocess Monitor operates under control of the TSC portion of the System Director (see paragraph 2.4.4).
- Off-line - In the off-line mode the Nonprocess Monitor does not time-share the computer, but operates as a dedicated monitor system under control of the Temporary Assembled Skeleton (see section 4).

3.1 NONPROCESS MONITOR OPERATIONS

Primarily, the function of the Nonprocess Monitor is to provide continuous control over a sequence of jobs that might otherwise require several individual systems. For example, the user may have a number of programs that are to be assembled or compiled, built into core loads, executed, and stored on disk for future use; and at the same time he may require that the running process be allowed to issue an interrupt that will be serviced immediately. The Nonprocess Monitor controls the sequencing of operations to load the Assembler or FORTRAN into core, to load the core load builder after the source program has been processed, to execute the object programs, etc. If an interrupt occurs during any of these operations, the Nonprocess Monitor relinquishes control to the System Director to handle that interrupt, after which control is returned to the Nonprocess Monitor if the amount of time allocated for time-sharing has not expired. (See figure 3-1.)

The off-line capability of the Nonprocess Monitor is necessary at system generation time, since programs must be assembled and stored on disk before the process control system can be generated. After system generation time if there is an occasion when the process is not running and, therefore, the computer is not needed to control it, the Nonprocess Monitor can be used to control computer operations, for example a data processing application such as a payroll program. (See figure 3-2.)

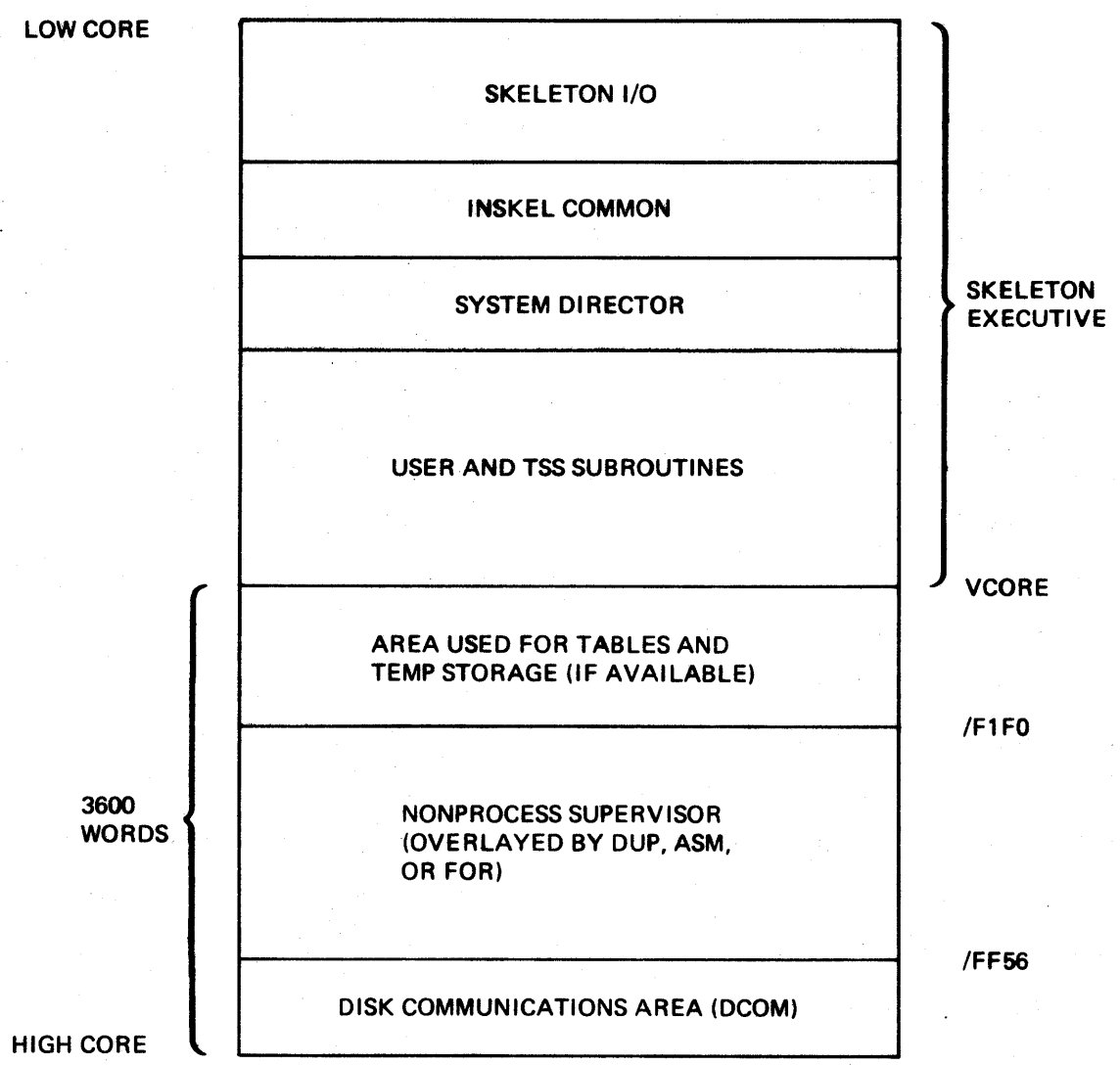


Figure 3-1. Nonprocess Monitor Storage (On-Line System)

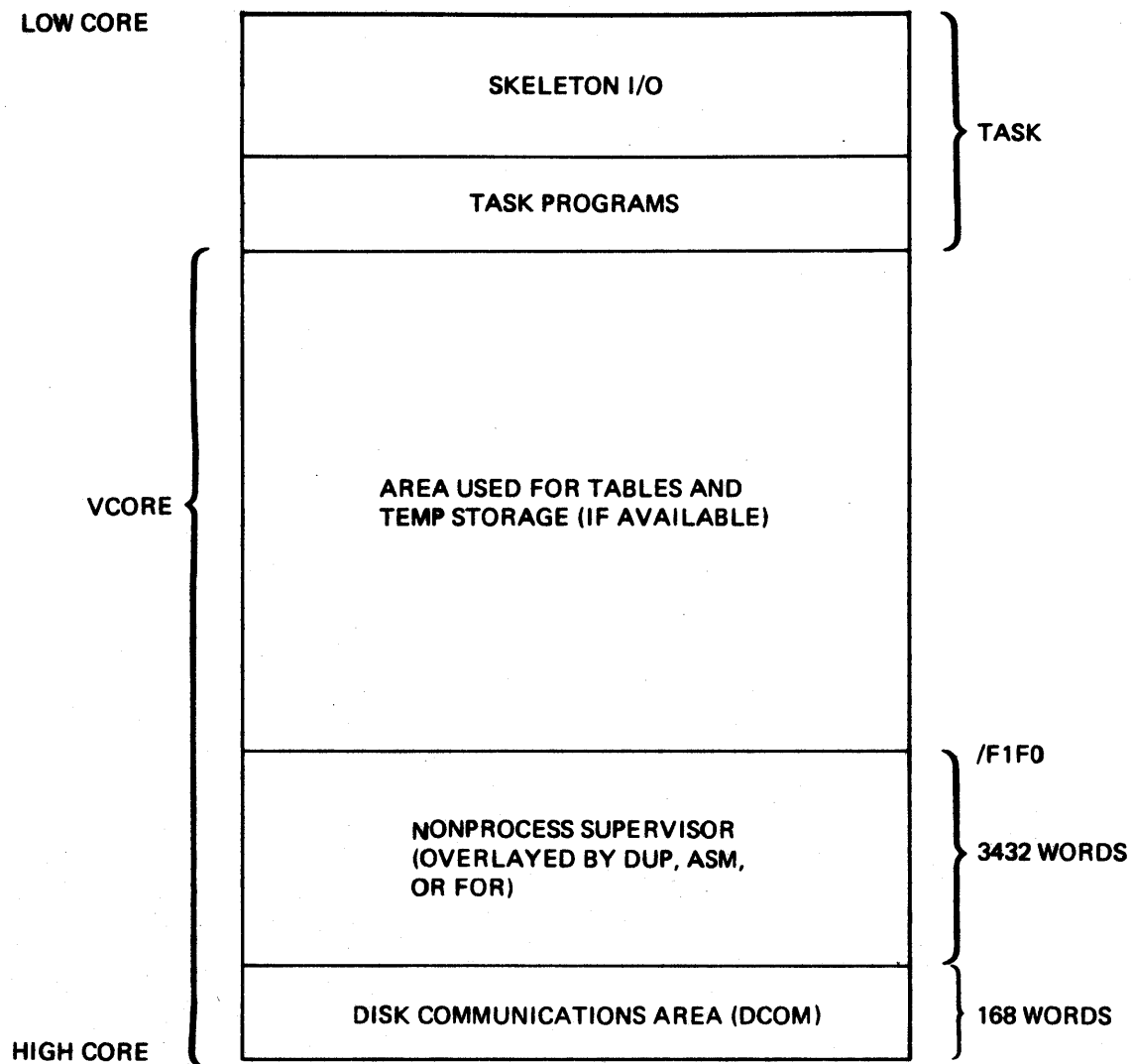


Figure 3-2. Nonprocess Monitor Storage (Off-Line System)

3.2 COMPONENTS OF THE NONPROCESS MONITOR

The Nonprocess Monitor comprises four programs as illustrated in figure 3-3. The functions of each of these components are described in the following paragraphs. The Nonprocess Monitor is a batch monitor that accepts card input and produces programs that may be stored on disc or executed. The monitor uses the Skeleton I/O routines (CARDN, DISKN, PRNTN, and TYPEN/WRTYN) for its input/output operations.

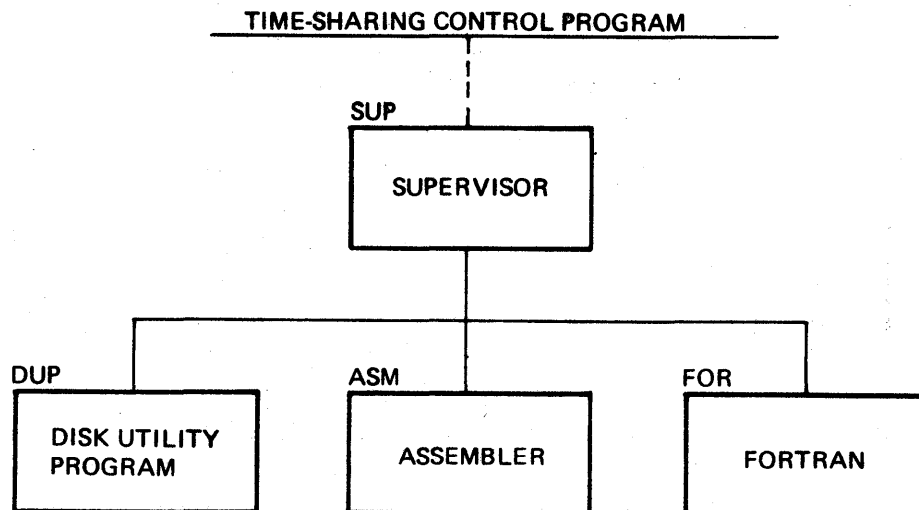


Figure 3-3. Nonprocess Monitor Components

3.2.1 Nonprocess Supervisor

The Nonprocess Supervisor (SUP) controls all Nonprocess Monitor operations. SUP consists of several routines, the two principal ones being the Skeleton Supervisor and the Monitor Control Analyzer.

The Skeleton Supervisor is read from disk into core whenever Nonprocess Monitor operation is initiated. It provides for communication between the Nonprocess Monitor components and user's programs that are being processed or executed. The Skeleton Supervisor provides for the orderly transfer of control from one program to another. Thus, several unrelated jobs may be stacked for processing and can be properly handled without operator intervention.

The Monitor Control Record Analyzer -- as its name signifies -- reads and interprets control records (such as //JOB, //ASM, //FOR) for the Nonprocess Monitor. This routine also outputs the records to the system printer and loads the appropriate monitor program for execution. The Monitor Control Record Analyzer accepts input from cards only. Each control record begins with two slashes and a blank. Thus, the control identifier begins in the fourth position. Table 3-1 lists the control records applicable to the Monitor Control Record Analyzer.

Table 3-1. Nonprocess Supervisor Control Records

Identifier	Function
// JOB	Causes termination of the previous job and initialization for the new one. When a job is aborted (e.g., if an unrecoverable I/O error occurs), cards in the card reader are bypassed until the next JOB card is encountered.
// END OF ALL JOBS	Indicates that there are no more nonprocess operations to be performed. It must be the last card in the input stream.
// ASM	Causes the Nonprocess Monitor to read the Assembler from disk into core storage for execution. The name of the mainline program to be assembled must be on this control card. Immediately following the ASM card must be the Assembler control cards, specifying the Assembler options (see paragraph 3.2.4), and the source language program. After a successful assembly, the object program is loaded as a temporary program.

Table 3-1. Nonprocess Supervisor Control Records (Cont.)

Identifier	Function
// FOR	Causes the Nonprocess Monitor to read the FORTRAN compiler from the disk into core storage for execution. The name of the main-line program to be compiled must be on this control card. Immediately following the FOR card must be the compiler control cards, specifying compiler options (see paragraph 3.2.3), and the FORTRAN language source program. After a successful compilation, the object program is loaded as a temporary program.
// DUP	Causes the Nonprocess Monitor to read the Disk Utility Program from disk into core storage for execution. Immediately following the DUP card must be the DUP control cards (see paragraph 3.2.2).
// XEQ	Causes the Nonprocess Monitor to load the named program and all required subroutines from disk into core storage for execution. This control card can also specify that a core load map be printed during the loading of a core load from relocatable programs.
// PAUS	Causes the Nonprocess Monitor to execute a WAIT instruction, allowing the operator to make setup changes. Monitor operation continues when the console STEP key is pressed. If an interrupt occurs during a wait period, it will be serviced, and control will be returned to the WAIT instruction.

Table 3-1. Nonprocess Supervisor Control Records (Cont.)

Identifier	Function
// *	Identifies a comment record. The contents of comment records are printed on the LIST and SYSTEM units.

3.2.2 Disk Utility Program

The Disk Utility Program (DUP) consists of a group of generalized routines that enable the user to perform easily the usual day-to-day operations of an installation. The Disk Utility Program is called by a // DUP control card, and DUP in turn calls its various routines, depending on the control statements that follow the // DUP card.

The DUP control statements begin with an asterisk in column 1. The code word that identifies the process to be performed appears in columns 2 through 10. Following column 10 information for the individual routine may be supplied. Table 3-2 lists the control statements that activate the individual Disk Utility Program routines. Table 3-3 lists control records that supply specific directions to these routines. (See also paragraph 5.10.)

3.2.3 FORTRAN Compiler

When a // FOR control record is read, the Nonprocess Monitor loads the FORTRAN compiler from disk into core storage, passes the name for the object program from the control record to the compiler, and transfers control to the compiler. The compiler reads the control cards and the source language statements that follow the // FOR control record. After compilation, the object program can be called for execution with an XEQ control card or can be stored on disk through a DUP operation. The FORTRAN compiler always generates object programs in relocatable format.

Table 3-2. Disk Utility Program Control Statements

Identifier	Function
*STORE	Stores relocatable programs in the relocatable program area on disk from cards or from the temporary program area. Parameters for this control statement include such information as the disk drive where the program is to be stored and the program name.
*STOREDATA	Stores a block of data on disk from cards or from the nonprocess work storage area. Parameters for this statement include identification of the disk drive and the name to be assigned to the data.
*STOREMOD	Allows the user to store core loads or modify existing nonprocess core loads and relocatable programs without previously deleting them.
*STORECI	Stores a program in core image form (i. e., as a core load) in the core load area and assigns a name to the core load. Parameters specify storage area, disk drive, type of core load, and map of core load area.
*DICLE	Allows an interrupt core load, which must already be on the disk, to be used to service more than one interrupt, e. g., a generalized error routine that handles spurious interrupts for all process core loads.
*DUMP	Writes programs from the disk to the specified device (cards or line printer) or from a program area to a nonprocess work storage area.

Table 3-2. Disk Utility Program Control Statements (Cont.)

Identifier	Function
*DUMPDATA	Writes data or a core image program from disk to a specified device (cards or line printer) or from a program area to a nonprocess work storage area.
*DUMPLET	Writes the contents of the location equivalence table (LET) or the fixed location equivalence table (FLET) to the line printer. The printout will include certain header information, alphabetic names, and disk addresses.
*DELET	Deletes the specified program, core load, or data file from the disk. A core load that is called by another core load or as the result of an interrupt can be deleted only if a replacement core load -- also specified via *DELET -- is available. When programs or data files are deleted, no checking is performed to determine whether they are referenced from any core loads.
*DEFINE	Defines variable parameter required by the system, such as number of disk drives, size of object core storage, packing of relocatable programs on disk, specific system programs to be removed from disk, and location and size of disk areas used by the system. This routine must be executed before the skeleton is built.

Table 3-2. Disk Utility Program Control Statements (Cont.)

Identifier	Function
*SEQCH	Enables the user to change the sequence in which mainline core loads are executed.
*DLABL	Places an identifying number in the first sector of the disk pack and writes sector addresses. If a nonsystem drive is specified, DLABL creates a LET area (defining the balance of the pack as an available area) starting in the second sector.
*DWRAD	Writes addresses within a specified area on the disk pack. Thus, the user can zero the data area, retain previous data, and initiate or remove file protection.

Table 3-3. Core Load Builder Control Records

Identifier	Function
*RCORD	Specifies the level and PISW bit positions for interrupts that are to be recorded if they occur during the execution of a core load. Only mainline and combination core loads need this control record.
*FILES	Establishes an equivalence between a symbolic file number used in a FORTRAN DEFINE FILE statement and the name in FLET of a data area or the disk drive for the data area. FILES can be used after an XEQ control record, a STORECI

Table 3-3. Core Load Builder Control Records (Cont.)

Identifier	Function
*INCLD	<p>control record, or a STOREMOD control record for a nonprocess core load.</p> <p>Specifies:</p> <ul style="list-style-type: none">● By name, level, and bit, the interrupt subroutines that are included with the mainline or combination core load.● The trace and error subroutines to be used with the mainline, interrupt, combination, or nonprocess core load.● By name and level, the programmed interrupt programs that are to be included with the mainline or combination core loads. <p>INCLD can be used after an XEQ control record, a STOREMOD control record, or a STORECI control record for a nonprocess core load.</p>
*LOCAL	<p>Identifies load-on-call subprograms. More than one program can be read into core by specifying several programs in one LOCAL block. Once the block is in core, it remains in the area until it is overlaid by another LOCAL. LOCAL can be used after an XEQ control record, a STOREMOD control record, or a STORECI control record for a nonprocess core load.</p>
*CCEND	<p>Signals the end of the loader control record stream. CCEND can be used after an XEQ control record (except when the XEQ record</p>

Table 3-3. Core Load Builder Control Records (Cont.)

Identifier	Function
	specifies a program in the core load area), a STOREMOD control record, a STOREDATA control record, or a STORECI control record for a nonprocess core load.
<p>Summary: These five control records enable the user to specify special core load requirements. FILES, LOCAL, INCLE, and CCEND are used to build nonprocess core loads. Process core loads (loaded with STORECI) permit all five records. Except for the CCEND record, all these control records can be multiples.</p>	

The compiler control cards begin with an asterisk in column 1, which is followed by a code word that identifies the process to be performed. Following the code word may be parameters required by the specific process. Table 3-4 lists the control cards applicable to the FORTRAN compiler.

3.2.4 Assembler

When a // ASM control record is read, the Nonprocess Monitor loads the Assembler from disk into core storage, passes the name for the object program from the control record to the Assembler, and transfers control to it. The Assembler reads the control cards and the source language statements that follow the // ASM control record. After the assembly the object program can be called for execution with an XEQ control card or can be stored on disk through a DUP operation. The Assembler can generate object programs in absolute or relocatable format.

Table 3-4. FORTRAN Control Records

Identifier	Function
*IOCS	Must be used to specify all I/O devices required for execution of the program, including all I/O devices used by any FORTRAN subprograms that are called. IOCS can appear only in the mainline program. The parameters (i. e., device names) must be separated by commas, and the list enclosed with parentheses.
*LIST SOURCE PROGRAM	Specifies that the source program is to be listed as it is read.
*LIST SUBPROGRAM NAMES	Specifies that the names of all subprograms (including EXTERNAL subprograms) called directly by the compiled program are to be listed.
*LIST SYMBOL TABLE	Specifies that all variable names, statement numbers, statement function names, and constants are to be listed along with their respective relative addresses.
*LIST ALL	Specifies that all three items (source program, subprogram names, and symbol table) are to be listed. If this record is used, the other three LIST records should not be included and vice versa.
*PUNCH	Causes control to be transferred automatically after a successful compilation to DUP to punch an object deck.
**	Causes the information from card columns 3 through 72 to be printed at the top of each page produced during compilation.

Table 3-4. FORTRAN Control Records (Cont.)

Identifier	Function
*TRANSFER TRACE	<p>Specifies that the compiler is to generate link-ages to a trace routine whenever an IF or Computed GO TO statement is encountered. Then, if data switch 15 is on at execution time, the trace output routine prints:</p> <ul style="list-style-type: none"> ● The expression of an IF statement, preceded by two asterisks. ● The value for the index of a Computed GO TO statement, preceded by three asterisks. <p>To select only parts of a program for tracing, the user places statements in the source program to indicate where tracing should start and stop:</p> <p style="padding-left: 40px;">CALL TSTRT (to start trace) CALL TSTOP (to stop trace)</p> <p>TRANSFER TRACE and the operation of data switch 15 are required as before, but now only the statements between TSTRT and TSTOP will be traced. (Data switch 15 can be turned off at any time to terminate tracing.)</p>
*ARITHMETIC TRACE	<p>Specifies that the compiler is to generate link-ages to a trace routine whenever an arithmetic statement is encountered. Operation is the same as for TRANSFER TRACE, except that the output routine prints the value for the assignment of a variable on the left of an equals sign</p>

Table 3-4. FORTRAN Control Records (Cont.)

Identifier	Function
	of an arithmetic statement, preceded by one asterisk.
*EXTENDED PRECISION	Directs the compiler to generate three-word real constants and real variables to provide extended precision for arithmetic operations.
*ONE WORD INTEGERS	Causes the compiler to allocate in the object program one word of storage for integer variables in a nonprocess program (instead of two words for standard precision or three words for extended precision). In a process program all integer variables are automatically generated as one word.
*NONPROCESS PROGRAM	Differentiates nonprocess programs from process programs. If this control record is not present in the source deck, the compiler assumes the program is a process program (and automatically forces one-word integer variables).

The Assembler control cards begin with an asterisk in column 1, which is followed by a code word that identifies the process to be performed. Following the code word may be parameters required by the specific process. Table 3-5 lists the control cards applicable to the Assembler.

Table 3-5. Assembler Control Records

Identifier	Function
*TWO PASS MODE	Allows the Assembler to produce the object program by performing two passes over the source program. Two passes are required when the nonprocess work storage area used by the Assembler is too small to hold the intermediate output of the assembly.
*LIST DECK	Directs the Assembler to output the uncompressed object program to punched cards. This operation requires the TWO PASS MODE control record. Errors are identified by codes punched in columns 18 and 19.
*LIST DECK E	Directs the Assembler to output punched cards only for those source statements that contain errors. This operation requires the TWO PASS MODE control record.
*LIST	Causes the Assembler to output a listing of the object program to the line printer. This operation requires the TWO PASS MODE control record.
*PUNCH	Directs the Assembler to output the compressed object program to punched cards. The card deck will be produced even if assembly errors are encountered.
*PUNCH SYMBOL TABLE	Causes the Assembler to output to punched cards the symbol table upon completion of the assembly.

Table 3-5. Assembler Control Records (Cont.)

Identifier	Function
*PRINT SYMBOL TABLE	Causes the Assembler to output the symbol table to the line printer upon completion of the assembly.
*SYSTEM SYMBOL TABLE	Directs the Assembler to incorporate the system symbol table as part of the assembly symbol table, thus enabling the user to reference system symbols without defining them in his own program.
*SAVE SYMBOL TABLE	Directs the Assembler to save the current assembly symbol table in the system symbol table area of disk storage, overlaying the previously saved symbol table.
*OVERFLOW SECTORS n	Indicates to the Assembler the number (n, where $1 \leq n \leq 32$) of sectors of nonprocess working storage available for possible symbol table overflow.
*COMMON n	Informs the Assembler that n words of the COMMON area are allotted for linkages between a FORTRAN mainline program and the assembled program.

SECTION 4 - SYSTEM EVOLVEMENT

The Time-Sharing Executive System provides the user a selection of operational modes. Through the Skeleton Executive he can perform process control and can time-share the computer via the Nonprocess Monitor. On the other hand, the user may choose to operate with the Nonprocess Monitor in an off-line mode. The user elects the option of constructing an on-line or off-line system at system generation time.

System generation is the process of preparing an operating system that conforms to the user-specified machine configuration and options. The process provides the facility for creation and maintenance of a monitored system that includes both supplied and user-written programs and subroutines. For the TSS system this facility is a stand-alone monitor program, the Temporary Assembled Skeleton.

4.1 TEMPORARY ASSEMBLED SKELETON

The Temporary Assembled Skeleton (TASK) enables the user to generate a system on disk from absolute and relocatable program decks that contain the executable phases and relocatable programs necessary for his installation. TASK is supplied as an Assembly-language source deck and, thus, is not directly usable by the installation. To assist the user in his initial system generation, General Automation supplies to each installation a "starter" program, called System Generation TASK.

4.2 SYSTEM GENERATION TASK

System Generation TASK (SYSGEN TASK) is supplied in assembled object format and contains the basic elements necessary for system generation: Nonprocess Monitor linkages, Skeleton Builder linkages, Absolute Loader. This starter program is a limited version of TASK and supports a minimum machine configuration:

- One GA 18/30 Industrial Supervisory System with a minimum of 8192 words of core storage
- One disk storage unit with one disk drive
- One card reader
- One card punch
- One ASR-33 Teletypewriter

4.3 SYSTEM GENERATION OVERVIEW

The process of generating a TSS system is described briefly in the following paragraphs to provide a point of orientation for the user in applying the procedures that are detailed in section 7. Individual programs and routines that are mentioned in these paragraphs and that have not been previously discussed are defined later in this section.

SYSGEN TASK is loaded into core storage by a four-card routine, called the TASK High Core Loader. After SYSGEN TASK is in core, the absolute loader function can be used to load the Disk-Write Addresses Program. This program initializes the disk to allow proper writing/reading of information. The system loader is then used to load the disk-resident programs, including the Disk Utility Program, Core Load Builder, and Nonprocess Monitor, to the disk. (Note that the System Director and TASK are supplied as source decks; other control programs are supplied in absolute format; and subroutines are in assembled, relocatable format.) Now, the Nonprocess Monitor can be called for execution to assemble the user's configuration of TASK.

After the Nonprocess Monitor is brought from disk into core storage for execution, the Nonprocess Supervisor accepts input from the card reader and calls the appropriate processor. Since TASK is supplied as an Assembly-language source deck, an ASM control card will direct the Supervisor to call the Assembler for execution. The user included with the TASK source deck certain control cards (i. e., EQU cards) that define the specific machine configuration and options for his installation. The Assembler assembles TASK and produces an object deck. Next, the System Director -- with user-defined options -- is assembled, and an object deck of it is produced. Finally, all user-written subroutines that are to be in the skeleton and the user-written program that is to be the initial (cold start) core load are assembled, and object decks for them are produced. At this point all the components required to construct the skeleton are available in assembled, object deck format.

The Skeleton Builder is the supplied routine that constructs the system skeleton. For an off-line system the skeleton consists of the TASK program, which is loaded to disk by the TASK Disk Loader, and the Skeleton I/O routines. For an on-line system the skeleton consists of the same Skeleton I/O routines that TASK uses, INSKELEL COMMON, the System Director, user-written and TSS subroutines, and an area for tables and control information needed by the System Director.

Once the TASK system skeleton has been built, it should be loaded into core, replacing SYSGEN TASK, for any further system generation functions. Operating the off-line system under TASK, the user can assemble or compile his programs that will execute under the Nonprocess Monitor. The next step in generating an on-line system is to build the core loads. The Nonprocess Monitor is called to assemble or compile the user-written process programs. Then the Core Load Builder forms the core loads and causes them to be written to disk in the proper format and with the necessary control information in the various tables (see glossary for definitions of Fixed Location Equivalence Table and Location Equivalence Table).

When the system is ready to operate, an on-line cold start is performed. The supplied cold start routine is loaded from cards. The user identifies his cold start core load to this routine via a name card. The cold start routine loads the Skeleton Executive into the skeleton area and directs it to load the cold start core load. The user's cold start core load is brought into VCORE and executed. Off-line execution is initiated through an off-line cold start. In this case, the user identifies TASK to the cold start routine via the name card. TASK is loaded into the skeleton area, and control is transferred to it. The user can then select which function to perform.

During system generation up to three separate disk packs can be produced. A non-defined pack, which is actually an intermediate step of system generation, exists with TASK in core and the supplied system on disk. This pack is used only for assembling and compiling operations. The Nonprocess Monitor pack contains TASK and the user's programs, data, etc. This is the off-line system with TASK

in the skeleton area and the Nonprocess Monitor in VCORE for execution of non-process programs. (Process code loads cannot be executed when the system is in off-line mode; i. e., with an off-line pack). The system pack contains the TSS system skeleton and the user's process core loads. Both process and nonprocess programs can be executed under time-sharing.

4.4 TASK DISK WRITE ADDRESSES PROGRAM

Among the object format routines supplied with the TSS system is the TASK Disk Write Addresses program. This program writes addresses on a specified disk and then checks each sector by reading and writing three different bit patterns a given number of times (specified by user). If no errors are encountered during this check, an appropriate message is output, and the disk is ready for use. If any errors are encountered during the check, a message is output, describing the sectors that are not acceptable. If too many sectors are defective, the disk pack is considered unacceptable, and an appropriate notice is output.

4.5 SYSTEM LOADER

The System Loader is used to store the TSS system on disk (the system disk drive). This program is supplied as an object deck which is loaded by the TASK Absolute Loader. The System Loader consists of five major components:

- System loader monitor
- Table builder
- Disk loader for absolute and relocatable programs
- Disk edit phase
- System loader error program

4.5.1 System Loader Monitor

This monitor controls the interface between the various components of the System Loader. It reads control cards, analyzes the information, and takes appropriate action. Furthermore, the monitor contains the library of input/output linkages called by all components of the System Loader.

4.5.2 Table Builder

After reading and verifying user-supplied assignment cards, the table builder routine develops the assignment and I/O unit tables and initiates the master branch table.

4.5.3 Disk Loader

Under control of the TASK Absolute Loader, the disk loader routine loads the TSS system to disk 0 and records the appropriate entries in the Location Equivalence Table (LET, see paragraph 4.8). Under control of the relocatable loader, this routine loads the supplied subroutines and records appropriate entries in LET for them.

4.5.4 Disk Edit Phase

This routine initializes the disk and the disk communications area with a standard format as a base for the TSS nonprocess programs.

4.5.5 System Loader Error Program

When an error is encountered during System Loader operation, the System Loader error program is called to analyze the error and to produce an error message.

Various types of errors are recognized, such as errors in control cards, procedural errors in the execution of the System Loader, and errors in the format or sequence of the loaded programs. The errors and recovery procedures are shown in appendix A.

4.5.6 System Loader Control Cards

There are two types of control cards the user must insert in the supplied system card deck prior to system generation:

- *DEDIT
- *ASSIGNMENT

These control cards enable the user to define the core size of the object system and to assign the system interrupt levels. Section 7 contains illustrations that show the proper placement of these control cards in the source deck.

4.5.6.1 Disk Edit Control Card

The disk edit control card (*DEDIT) is supplied partially pre-punched; it must be completed by the user and included with the system deck at system generation time. The *DEDIT is used to define the core size of the object machine and the disk message buffer size. This card is the last control card read by the System Loader. It causes the System Loader to initialize the Fixed Location Equivalence Table (see paragraph 4.8) area on the disk, to calculate the core size of the source machine, to record the core size of the object machine in the disk communications area, to provide file protection of the system and subroutine area, and to return control to TASK.

The format of the *DEDIT card is

1	8	12	15
*DEDIT	ddK	nnn	CYL

- *DEDIT - identifies this card as the disk edit control card.
- dd - two-digit, decimal number that specifies the core size of the object machine. One of three values may be punched in these columns: 08, 16, 32. Any other value is invalid.
- K - indicates that the preceding two-digit number is in thousands; i.e., 8K, 16K, or 32K core size.
- nnn - the number of groups of eight sectors to be used for the object-time disk message buffer. The entry must be in the range: $000 \leq nnn \leq 199$.
- CYL - the letters CYL must be punched in columns 15 through 17.

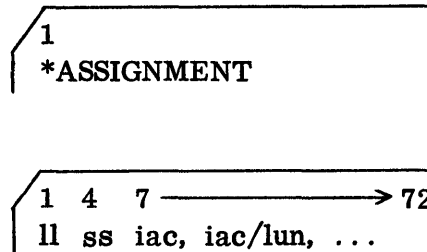
The remaining card columns must be blank.

4.5.6.2 Assignment Cards

The assignment cards allow the user to assign I/O devices and machine functions to particular interrupt levels. These assignments are made in the form of interrupt assignment codes and logical unit numbers. Interrupt assignment codes (IAC) are

fixed for each device and cannot be changed. Logical unit numbers (LUN) are selected by the user for linkage to FORTRAN programs. The permitted values for LUNs are 01 through 44. Each device in the system must have one unique LUN assigned to it. The fixed interrupt assignment codes and permissible LUN values for the TSS system are listed in table 4-1.

The formats of the assignment cards are:



- *ASSIGNMENT - this card is supplied with the system deck; it contains only these 11 characters and precedes the user-prepared assignment cards.
- 11 - specifies the interrupt level to which this card is applicable. Each interrupt level used must be specified on an assignment card. The entry must be in the range of $00 \leq 11 \leq 23$ or 99. The entry 99 specifies a dummy level to provide FORTRAN linkages for either the keyboard-printers or second magnetic tape unit.
- ss - designates the number of interrupt level status bits assigned to this level (ILSW). The entry must be in the range $01 \leq ss \leq 16$ or CC. The letters CC indicate that this is a continuation card; i. e., columns 7 through 72 of the preceding card were not sufficient to contain all the ILSW bit assignments and, therefore, the assignments are continued on this card.

Table 4-1. Interrupt Assignment Code/Logical Unit Number Assignments

Device	IAC (Decimal)	Permitted LUN Values
Interval timers	00	No LUN assignable
First keyboard-printer or TTY on printer group 1	01	01 through 44
Card reader	02	01 through 44
Paper tape reader	03	01 through 44
First disk unit	04	No LUN assignable
Plotter 1	05	01 through 44
Line printer 1	06	01 through 44
Reserved	07	No LUN assignable
Second disk unit	08	No LUN assignable
Third disk unit	09	No LUN assignable
First ADC, analog input basic	10	No LUN assignable
Digital input	11	No LUN assignable
Digital analog output	12	No LUN assignable
Channel adaptor	13	No LUN assignable
Magnetic tape unit	14	01 through 44
First keyboard-printer or TTY on printer group 2	15	01 through 44
Analog input expander, second ADC	16	No LUN assignable
Card punch	17	01 through 44
Line printer 2	18	01 through 44
Plotter 2	19	01 through 44

Table 4-1. Interrupt Assignment Code/Logical Unit Number Assignments (Cont.)

Device	IAC (Decimal)	Permitted LUN Values
Keyboard 2	20	01 through 44
Keyboard 3	21	01 through 44
Keyboard 4	22	01 through 44
Keyboard 6	23	01 through 44
Keyboard 7	24	01 through 44
Keyboard 8	25	01 through 44
Paper Tape Punch	26	01 through 44
Command reject INT (magnetic tape)	27	No LUN assignable
End of table INT (magnetic tape)	28	No LUN assignable
Reserved	29-31	No LUN assignable
Console interrupts	32	No LUN assignable
Process interrupts	33	No LUN assignable
First comparator	34	No LUN assignable
Second comparator	35	No LUN assignable
Second TTY on printer group 1	36	01 through 44
Third TTY on printer group 1	37	01 through 44
Fourth TTY on printer group 1	38	01 through 44
Second TTY on printer group 2	39	01 through 44
Third TTY on printer group 2	40	01 through 44
Fourth TTY on printer group 2	41	01 through 44

Table 4-1. Interrupt Assignment Code/Logical Unit Number Assignments (Cont.)

Device	IAC (Decimal)	Permitted LUN Values
Keyboard 1 [†]	42	01 through 44
Keyboard 5 [†]	43	01 through 44
Magnetic tape drive 2 [†]	44	01 through 44
RPQX ^{††}	45-63	No LUN assignable
<p>[†]Require interrupt level 99 (see paragraph 4.6.2).</p> <p>^{††}RPQX is an extension of RPQ for the disposition of the user.</p>		

iac, iac/lun ... - groups of IAC or IAC/LUN assignments. The number of groups must correspond to the number specified by ss (columns 4-5). Groups are separated by commas. Each group can consist of an IAC or an IAC/LUN combination. When a group contains both an IAC and a LUN, the two values must be separated by a slash. Not all IACs have LUNs assigned. When an IAC has an assignable LUN, but the LUN is not specified on the assignment card, the System Loader assigns the LUN the same number as the IAC. A LUN number can be assigned to only one device.

Card columns 12 through 80 of the *ASSIGNMENT card and columns 3, 6, and 73 through 80 (as well as any unused columns from 7 through 72) of the individual assignment cards must be blank.

The System Loader builds a table of assignments, providing space for all LUNs up to the highest LUN assigned. Therefore, when a system has minimum core storage (8K system), the user should assign consecutive LUN numbers beginning with the lowest possible value to limit the size of the table.

4.5.6.3 Comment Cards

The user has the option of including comment cards anywhere in the system deck. The contents of these cards have no effect on system generation, but are merely output to the printer.

The format of a comment card is:

```
  1 2 3 5  
  / / * any character string
```

The slashes in card columns 1 and 2 identify the comment card to the System Loader. Card column 3 must contain a nonblank character. The comments begin in column 5 and are terminated in column 72 or by three consecutive blank columns - whichever occurs first.

4.6 CORE LOAD BUILDER

As explained in Section 2, executable programs must be segmented into units called core loads. This process is accomplished by the Core Load Builder. This program operates under control of the Nonprocess Monitor. User-supplied control records provide the Core Load Builder with information, such as the names of the relocatable mainline, interrupts to be recorded, data files to be used, interrupt routines to be included as part of the core load, and LOCAL subprograms. Using this information along with information provided by the System Loader and the Skeleton Builder, the Core Load Builder generates the tables, transfer vectors, and work areas that are combined with the instructions that make up a core load. The Core Load Builder is used to construct process mainline, interrupt, and nonprocess core loads for storage in the core image area on disk.

4.7 SKELETON BUILDER

The Skeleton Builder is the program that actually constructs the system skeleton from the user-written and supplied routines. These routines must have been assembled or compiled and stored on disk in relocatable format before the Skeleton Builder is executed. The Skeleton Builder constructs the system skeleton in core image format and stores it on disk; thereafter, the skeleton can be read into core, for execution, by a cold start procedure.

In addition to its function at the initial system generation, the Skeleton Builder is used to rebuild the system skeleton. Rebuilding the system skeleton is necessary any time routines are added, deleted, or modified.

4.8 DISK LOCATION EQUIVALENCE TABLES

Two tables are maintained by the system to record information concerning the location of programs, data, and core loads that are stored on disk. These tables are the Location Equivalence Table and the Fixed Location Equivalence Table.

The Location Equivalence Table (LET) provides a map of system programs, sub-routines, and relocatable programs on disk. Each entry in LET occupies three words

and includes the name of the function and its size (i.e., disk block count, where one disc block is 20 words):

NAME	DISK BLOCK COUNT
------	---------------------

An entry is made in LET for each entry point in a subroutine. As user-written relocatable programs are stored on disk, entries for them are also made in LET.

The System Loader loads the TSS programs for system operation. From information supplied to it through control cards, the System Loader makes initial entries in LET concerning the various system programs (FORTRAN, subroutines, skeleton, etc.), work areas, save areas, message buffer area, branch tables, disk communication area, core image program area, and process cold start programs.

Entries are also made in LET by DUP. Following the assembly or compilation of a program, the relocatable object program is stored on disk in the relocatable program area, and the appropriate entries are recorded in LET.

The Fixed Location Equivalence Table (FLET) provides a map of core loads and data stored in the process core image storage (or core load) area and the various save areas on disk. Entries in FLET occupy four words:

NAME	WORD COUNT	SECTOR ADDRESS
------	---------------	-------------------

Entries are made in FLET by the Core Load Builder after it converts relocatable programs to core loads.

When a program or core load is deleted, its name in LET or FLET is replaced by the name 9DUMY. Since the system can no longer locate the program or core load name in the table (LET or FLET), that program or core load cannot be referenced. Therefore, the area on disk that the program or core load had occupied is available for the storage of other programs, core loads, or data files.

4.9 COLD START ROUTINE

The first routine to be executed by the system is the Cold Start routine. This supplied routine is loaded with the system deck and resides in the storage protected skeleton area on disk. Cold Start operation is initiated by a two-card cold start loader and a name card. (Illustrations of these cards and procedures for their use are given in section 7.) At least one process core load must be on disk in the core load area to be called to start the system.

The two-card cold start loader is supplied in a ready-to-use format; however, the name card must be punched by the user. Its format is

```
1      8      14 16 18 20 22 24 → 80
*CLDST name p  c  d1 d2 d3 comments
```

- *CLDST - must be punched in columns 1 through 6.
- name - identifies the first core load to be called. This entry is required. The name is entered in the field left justified, may consist of up to five characters, and must begin with an alphabetic character.
- p - storage protection option:
 - blank = no storage protection
 - 1 = storage protection selected
- c - clock option:
 - blank = no option selected
 - 1 = clock option selected

d_1, d_2, d_3 - logical disk drive assignment ($d_1 \neq d_2 \neq d_3$):

d_1 = required; assigns a physical drive number to logical drive 0. It must be punched 0, 1, or 2.

d_2 = optional; assigns a physical drive number to logical drive 1. It may be blank or punched 0, 1, or 2.

d_3 = optional; assigns a physical drive number to logical drive 2. It may be blank or punched 0, 1, or 2.

comments - any comments may be entered in columns 24 through 80.

Columns 7, 13, 15, 17, 19, 21, and 23 must be blank.

The core load named in columns 8 through 12 is entered in full mask mode, i. e., all interrupt levels masked. It is the user's responsibility to unmask for his system configuration.

The nonprocess TASK name card has the same entries as described above with these exceptions:

- name (columns 8 through 12) must be TASK.
- p and c (columns 14 and 16) must be blank.

TASK will unmask all levels.

4.10 TASK EQUATE CARDS

During the system generation procedure, the user must define the specific configuration of TASK for his installation. This definition is accomplished through the use of Assembler-language equate (EQU) cards. After preparing the EQU cards, the user inserts them into the TASK source deck. (Section 7 contains illustrations that show the proper location of the EQU cards in the source deck.) Thus, the assembled TASK will be tailored for the specific hardware configuration and process requirements.

The TASK equate cards are arranged in two groups and are referred to as "group 1 and group 2 EQU cards." There are certain rules concerning the use of EQU cards that must be followed:

1. All EQU cards for both group 1 and group 2 must be included in the source deck when TASK is assembled.
2. The entries on the cards must be left justified in their respective fields.
3. If an EQU card is not applicable to the configuration being defined, that EQU card must contain a 0 in column 35.

Tables 4-2 and 4-3 define the group 1 and group 2 EQU cards, respectively. These tables specify the information to be supplied and the card columns it is to occupy.

4.11 SYSTEM DIRECTOR EQUATE CARDS

As with the TASK program, the System Director must be assembled during system generation. There are a number of EQU cards the user must provide to ensure a successful assembly. Section 7 contains illustrations that show the proper placement of these control cards in the source deck. Table 4-5 defines the information to be supplied and the card columns it is to occupy.

Certain groups of these equate cards are interdependent. That is, the values given to the groups

NIL00 through NIL23

USE00 through USE23

NB00 through NB23

and NULEV must not conflict. The following rules apply to the use of System Director equate cards:

1. The System Director NULEV and the TASK NULEV equate cards must contain the same value.
2. The value of NULEV must be greater than the value of these TASK equate cards: CONTA, LVPRI, TYPL1, TYPL2.

3. The value of NULEV must be 1 greater than the highest number USExx equate cards that contains a 1; e.g., if USE16 is the highest numbered USExx card, NULEV would be 17.
4. USExx equate cards that are assigned the value 1 must be consecutively numbered starting with USE00.
5. NBxx equate cards must contain 0 when the corresponding USExx card is 0, or must contain a value between 1 and 16 when the corresponding USExx card is 1.
6. The value of the NLWS1 equate card plus that of the NLWS2 card must be equal to or less than the value of NULEV.
7. The sum of the values of NITP1 and NITP2 must be 0 if the value of ITCUS is 0.

Table 4-2. Group 1 TASK EQU Cards

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
BZ1	EQU	20-319	Message unit size of TTY 1-1. [†]
BZ2	EQU	20-319	Message unit size of TTY 1-2. [†]
BZ3	EQU	20-319	Message unit size of TTY 1-3. [†]
BZ4	EQU	20-319	Message unit size of TTY 1-4. [†]
BZ5	EQU	20-319	Message unit size of TTY 2-1. [†]
BZ6	EQU	20-319	Message unit size of TTY 2-2. [†]
BZ7	EQU	20-319	Message unit size of TTY 2-3. [†]
BZ8	EQU	20-319	Message unit size of TTY 2-4. [†]
CDINS	EQU	0, 1	0 = CARDN is not to be in Skeleton I/O. ^{††} 1 = CARDN is to be in the Skeleton I/O. ^{††}
COMSZ	EQU	$x \geq 0$	Maximum size of INSKEL COMMON for object machine. The value of x may be zero or any positive decimal number that does not cause the skeleton size to exceed the VCORE boundary.
CONTA	EQU	0-23	Interrupt level for console interrupt routine programming when a console interrupt occurs and data switch 7 is off.
CORSZ	EQU	8, 16, 32	Core size of object machine.

[†]If the TTY has not been defined as buffering messages to disk and FORTRAN compilations are planned, the message unit size must be greater than 80.

^{††}When CARDN is to be in the skeleton, it must be included in the Skeleton I/O; it cannot be referenced in an *INCLD card at skeleton build time. If a Nonprocess Monitor pack is required, the user should equate CDINS to 1 to conserve space.

Table 4-2. Group 1 TASK EQU Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
CRDNO	EQU	1	TSS system requires both a card reader and a card punch; therefore, a 0 value is not applicable.
DORG1	EQU	0, 1	0 = disk 1 on system. 1 = Any other configuration.
DORG2	EQU	0, 1	0 = More than one disk drive on system. 1 = 2311 disk drive.
ECPT1	EQU	0, 1	0 = EAC [†] printer is TTY. 1 = EAC [†] printer is line printer.
ECPT2	EQU	1-15	If EAC [†] printer is a TTY, see table 4-4 for possible combinations.
ECPT3	EQU	0, 1	0 = EAC printer is TTY printer group 1. 1 = EAC printer is TTY printer group 2.
INTKY	EQU	1-23	User's interrupt level for TYPEN routine programming when a teletypewriter interrupt occurs. Value must be greater than that for TYPL1 and/or TYPL2.
LORG1	EQU	0, 1	0 = LIST printer is a TTY. 1 = LIST printer is a line printer.
LVPR1	EQU	0-23	Interrupt level of line printer.

[†]EAC stands for TASK Error Alert Control.

Table 4-2. Group 1 TASK EQU Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
MKLEV	EQU	0, 1	<p>0 = 14 or fewer interrupt levels on system.</p> <p>1 = 15 or more interrupt levels on system.</p>
NOBUF	EQU	0, 1	<p>0 = No buffering of messages to disk.</p> <p>1 = Buffering of messages to disk.</p>
NOCYL	EQU	1-200	<p>Number of groups of eight sectors on disk for buffering of messages to the TTY.</p>
NULEV	EQU	1-24	<p>Number of interrupt levels for final TSS system; e. g. , if interrupt levels 0-6 are used, the NULEV value is 7.</p>
NUMBE	EQU	1-1600	<p>Maximum number of disk sectors that can hold nonprocess messages at any one time. This area may also be used for process messages. The number of sectors must not exceed NOCYL x 8.</p>
ONLIN	EQU	0, 1	<p>0 = Delete the absolute loader and the skeleton builder from TASK. The resulting TASK deck is to be used only for execution of the Nonprocess Monitor and not for TSS system generation; i. e. , gives the user an off-line system that provides maximum core for execution of Nonprocess Monitor programs.</p> <p>1 = Provide all TASK functions.</p>

Table 4-2. Group 1 TASK EQU Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
ORLP1	EQU	0	No overlap on analog input basic.
ORLP2	EQU	0	No overlap on analog input expander.
PORG	EQU	0, 1	0 = No line printer on system. 1 = Any other configuration.
PRICS	EQU	0, 1	0 = Standard precision arithmetic subroutines for process programs. 1 = Extended precision arithmetic subroutines for process programs.
PRIL0	EQU	0-23	Interrupt level of 2311 disk drive.
PTSKP	EQU	0, 1	0 = Transfer to EAC [†] for all line printer not-ready errors. 1 = For a not-ready error when PRNTN has been called by a nonprocess program, loop on not-ready; otherwise, branch to EAC. [†]
SLORG	EQU	1-8	If the SYSTEM or LIST printer is a TTY (see SORG1 and LORG1), it can be any on the system. If both SYSTEM and LIST printers are TTYS, they must be assigned to the same TTY.
[†] EAC stands for TASK Error Alert Control.			

Table 4-2. Group 1 TASK EQU Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
SORG1	EQU	0, 1	0 = SYSTEM printer is a TTY. 1 = SYSTEM printer is a line printer.
TA01	EQU	0, 1	0 = Delete full trace from utility package. 1 = Retain full trace in utility package.
TA02	EQU	0, 1	0 = Delete check/stop trace from utility package. 1 = Retain check/stop trace in utility package.
TA03	EQU	0, 1	0 = Delete disk dump from utility package. 1 = Retain disk dump in utility package.
TORG	EQU	0, 1	0 = No TTY on the system. 1 = Any other configuration.
TORG1	EQU	0, 1	0 = One TTY on printer group 1. [†] 1 = Any other configuration.
TORG2	EQU	0, 1	0 = Two TTYs on printer group 1. [†] 1 = Any other configuration.
TORG3	EQU	0, 1	0 = Three TTYs on printer group 1. [†] 1 = Any other configuration.
TORG4	EQU	0, 1	0 = No keyboard-printer on printer group 1. 1 = A keyboard-printer on printer group 1

[†]Count a keyboard-printer as if it were a TTY.

Table 4-2. Group 1 TASK EQU Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
TORG5	EQU	0, 1	0 = No keyboard-printer/TTY on printer group 2. 1 = Any other configuration.
TORG6	EQU	0, 1	0 = One TTY on printer group 2. [†] 1 = Any other configuration.
TORG7	EQU	0, 1	0 = Two TTYS on printer group 2. [†] 1 = Any other configuration
TORG8	EQU	0, 1	0 = Three TTYS on printer group 2. [†] 1 = Any other configuration.
TORG9	EQU	0, 1	0 = No printer-keyboard on printer group 2. 1 = A printer-keyboard on printer group 2.
TORGN	EQU	0, 1	0 = No keyboard-printer/TTY on system. 1 = Any other configuration.
TRORG	EQU	0, 1	0 = Utility package is not to be in TASK (when this value is 0, TA01, TA02, and TA03 are ignored). 1 = Utility package is to be in TASK.
TYPL1	EQU	0-23	Interrupt level for TTY printer group 1. ^{††}

[†]Count a keyboard-printer as if it were a TTY.

^{††}If a keyboard-printer has been included in the group, interrupt level 23 is not a valid assignment for the group.

Table 4-2. Group 1 TASK EQU Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Values (cc: 35-71)	Meaning
TYPL2	EQU	0-23	Interrupt level for TTY printer group 2. [†]
[†] If a keyboard-printer has been included in the group, interrupt level 23 is not a valid assignment for the group.			

Table 4-3. Group 2 TASK EQU Cards

X identifies the backup printer assigned to the printer on the equate card ECPT2. If no backup unit is available, assign the X value of the printer being equated. The values of X are:

<u>X</u>	<u>TTY Device</u>	<u>X</u>	<u>TTY Device</u>
DT1	TTY 1-1	DT5	TTY 2-1
DT2	TTY 1-2	DT6	TTY 2-2
DT3	TTY 1-3	DT7	TTY 2-3
DT4	TTY 1-4	DT8	TTY 2-4

Label (cc: 21-25)	Operation Code (cc: 27-30)	Value (cc: 35-71)	Meaning
BDT1	EQU	X	Backup TTY for TTY 1-1.
BDT2	EQU	X	Backup TTY for TTY 1-2.
BDT3	EQU	X	Backup TTY for TTY 1-3.
BDT4	EQU	X	Backup TTY for TTY 1-4.
BDT5	EQU	X	Backup TTY for TTY 2-1.
BDT6	EQU	X	Backup TTY for TTY 2-2.
BDT7	EQU	X	Backup TTY for TTY 2-3.
BDT8	EQU	X	Backup TTY for TTY 2-4.

Note: Count a keyboard-printer as if it were a TTY.

Table 4-4. Error Alert Control Printer Combinations

EQU Value	TTY 1-1 or 2-1	TTY 1-2 or 2-2	TTY 1-3 or 2-3	TTY 1-4 or 2-4
1	Yes	No	No	No
2	No	Yes	No	No
3	Yes	Yes	No	No
4	No	No	Yes	No
5	Yes	No	Yes	No
6	No	Yes	Yes	No
7	Yes	Yes	Yes	No
8	No	No	No	Yes
9	Yes	No	No	Yes
10	No	Yes	No	Yes
11	Yes	Yes	No	Yes
12	No	No	Yes	Yes
13	Yes	No	Yes	Yes
14	No	Yes	Yes	Yes
15	Yes	Yes	Yes	Yes

Table 4-5. System Director Equate Cards

Label (cc: 21-25)	Operation Code (cc: 27-30)	Permissible Values (cc: 35-71)	Meaning
CBASE	EQU	XXXXXX	The number of times the time clock is to be incremented before the programmed times are incremented.
DUMP1	EQU	0, 1	0 = The routine that dumps core to disk is not to be included. 1 = The routine that dumps core to disk is to be included.
ICLL1 ICLL2	EQU EQU	/XXXX /XXXX	These two cards define two 16-bit words that are used to identify the interrupt levels the user has elected to mask for the servicing of out-of-core interrupts. The first 12 bit positions of the ICLL1 value correspond to the 12 standard interrupt levels (0 through 11) on the system. Bit positions 13 and 14 of ICLL1 and bits 0 through 10 of ICLL2 correspond to the additional 12 levels available on the system. The contents of the defined bit positions of these words are stored in the interrupt mask register.
ITCUS	EQU	0, 1	0 = The ITC program is not to be included in the System Director. 1 = The ITC program is to be included in the System Director.
NB00 NB01 NB02 NB03 NB04 NB05	EQU EQU EQU EQU EQU EQU	XX XX XX XX XX XX	00 ≤ XX ≤ 16. Labels NB00 through NB23 are equated to the rightmost bit plus 1 that is assigned to an ILSW for a level. If there are no bits on a level, the label must be equated to 0.

Table 4-5. System Director Equate Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Permissible Values (cc: 35-71)	Meaning	
NB06	EQU	XX		
NB07	EQU	XX		
NB08	EQU	XX		
NB09	EQU	XX		
NB10	EQU	XX		
NB11	EQU	XX		
NB12	EQU	XX		
NB13	EQU	XX		
NB14	EQU	XX		
NB15	EQU	XX		
NB16	EQU	XX		
NB17	EQU	XX		
NB18	EQU	XX		
NB19	EQU	XX		
NB20	EQU	XX		
NB21	EQU	XX		
NB22	EQU	XX		
NB23	EQU	XX		
NIL00	EQU	XX		<p>00 ≤ XX ≤ 16. The NIL00 through NIL23 equate cards define PSIW1 through 24 respectively. XX equals the value of 1 plus the highest numbered PISW bit assigned to a process interrupt; XX equals 0 if no process interrupts are assigned to a level. For multiple PISWs the NILxx values should only be assigned to those PISWs that are not serviced as I/O devices.</p>
NIL01	EQU	XX		
NIL02	EQU	XX		
NIL03	EQU	XX		
NIL04	EQU	XX		
NIL05	EQU	XX		
NIL06	EQU	XX		
NIL07	EQU	XX		
NIL08	EQU	XX		
NIL09	EQU	XX		
NIL10	EQU	XX		
NIL11	EQU	XX		
NIL12	EQU	XX		
NIL13	EQU	XX		
NIL14	EQU	XX		
NIL15	EQU	XX		
NIL16	EQU	XX		
NIL17	EQU	XX		

Table 4-5. System Director Equate Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Permissible Values (cc: 35-71)	Meaning
NIL18	EQU	XX	
NIL19	EQU	XX	
NIL20	EQU	XX	
NIL21	EQU	XX	
NIL22	EQU	XX	
NIL23	EQU	XX	
NITP1	EQU	XX	<p>$1 \leq XX \leq 16$. Number of CALL COUNT subroutines 0-15. This value is 1 plus the highest numbered subroutine in the first group (0-15).</p>
NITP2	EQU	XX	<p>$1 \leq XX \leq 16$. Number of CALL COUNT subroutines 16-31. This value is 1 plus the highest numbered subroutine in the second group [16-31 = (0-15) + 1].</p>
NLWS1	EQU	XX	<p>$1 \leq XX \leq 14$. The value 1 plus the number of the lowest priority level (0-13) assigned to a programmed interrupt.</p>
NLWS2	EQU	XX	<p>$1 \leq XX \leq 10$. The value 1 plus the number of the lowest priority level (14-23) assigned to a programmed interrupt.</p>
NULEV	EQU	XX	<p>$1 \leq XX \leq 24$. Specifies the number of interrupt levels to be compiled in the System Director. The value is 1 plus the highest numbered interrupt level used. If levels 0 through 9 are used, NULEV is equated to 10 (also see USExx equate card).</p>

Table 4-5. System Director Equate Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Permissible Values (cc: 35-71)	Meaning
NUQUE	EQU	XX	Specifies the number of entries to be allowed in the Queue Table. This number should allow sufficient storage so that the Queue Table will not overflow under normal conditions. Each entry point in the Queue Table requires three words of storage.
OPMO1	EQU	0, 1	0 = The user is to reset the operations monitor. 1 = The operations monitor is to be reset by ITC when time-sharing is in progress.
TBASE	EQU	-XXXXX	This negative value is put in timer C (word 6) to be counted down for the time clock base.
TIMES	EQU	0, 1	0 = Time-sharing is not to be used. 1 = Time-sharing is to be used.
TIME1 TIME2	EQU EQU	/XXXX /XXXX	The labels TIME1 and TIME2 are the hexadecimal equivalence of a double-precision number that defines the time in milliseconds that is calculated by the equation: $\text{TBASE} * \text{HARDWARE BASE}$ for timer C TIME1 is set to 10000 except when the calculated time exceeds 65,535 milliseconds, in which case two words are required.

Table 4-5. System Director Equate Cards (Cont.)

Label (cc: 21-25)	Operation Code (cc: 27-30)	Permissible Values (cc: 35-71)	Meaning
TISHA	EQU	XXXXXX	The number of times the programmed clock is to be updated before time-sharing is terminated.
USE00 USE01 USE02 USE03 USE04 USE05 USE06 USE07 USE08 USE09 USE10 USE11 USE12 USE13 USE14 USE15 USE16 USE17 USE18 USE19 USE20 USE21 USE22 USE23	EQU EQU	0, 1 0, 1	Labels USE00 through USE23 define the level work areas. A label is equated to 0 if no work area is included for that level; it is equated to 1 if a work area is included on that level. When USE14 is equated to 0, the XIO and other coding for levels 14 through 23 will be originated out.
VCORE	EQU	XXXXXX	The starting address of variable core; must be an even number.

SECTION 5 – PROGRAMMING CONSIDERATIONS

This section contains examples and programming sequences to illustrate the recommended approach for various situations.

5.1 FORTRAN SUBPROGRAMS

Often the same group of statements is required to be executed at various locations throughout a program. In order to speed up and simplify coding the subprogram is used. A subprogram consists of a set of logical steps to perform a predefined operation on data supplied to it by the calling program. This data is supplied by the use of arguments.

In 18/30 TSS FORTRAN there are two types of subroutines: SUBROUTINE and FUNCTION subprograms. In each case the subprogram is coded and compiled only once. It is then executed as needed without being included in the in-line coding.

5.1.1 SUBROUTINE Subprograms

The first statement in a SUBROUTINE subprogram is the SUBROUTINE statement:

SUBROUTINE namex (a, b, c,)

namex name of the subprogram; it must be one to five alphanumeric characters, and the first character must be alphabetic.

a, b, c, argument list. These consist of the variables to be used in the subprogram. They may be nonsubscripted variables, array names, or another subprogram name.

This is a dummy argument list so the names do not have to agree with those in the calling program. However, the arguments must agree in number, type (REAL or INTEGER), order, and array size. None of the arguments may appear in an EQUIVALENCE statement in the subprogram. When an argument is an array name, the argument must appear in a DIMENSION statement in the subprogram.

To establish the linkage to a SUBROUTINE subprogram, the CALL statement is used:

```
CALL namex (x,y,z,....)
namex      symbolic name of the SUBROUTINE subprogram
x,y,z,.... arguments being supplied to the subprogram
```

The arguments in a CALL statement may be real or integer type and may be subscripted variables, constants, array names, subprogram names, or arithmetic expressions. The only restriction is that they must agree in order, type, and array size with the corresponding arguments defined in the subprogram.

Example:

```
// JOB      X      Y      Z
// FOR SUBR
SUBROUTINE SUBR(A,I,ARRAY)
DIMENSION ARRAY(10)
.
.
ARRAY(I) = A
.
.
RETURN
END
// FOR PROGM
DIMENSION X(10)
.
.
Y = 40.075
.
.
CALL SUBR(Y,6,X)
.
.
END
// DUP
*STORECI M          PROGM PROGM RSTRT
*CCEND
// JOB      X      Y      Z
// END OF ALL JOBS
```

5.1.2 FUNCTION Subprograms

The FUNCTION subprogram is a FORTRAN subroutine that is executed whenever its name appears in a FORTRAN program. The rules for FUNCTION names are the same as those for FORTRAN variables.

The first statement in a FUNCTION subprogram is the FUNCTION statement:

```
FUNCTION namex (i,j,k....)
```

```
namex          symbolic name of the subprogram
```

```
i,j,k....     arguments to be passed to the subprogram
```

The rules for arguments in a FUNCTION subprogram are the same as those for a SUBROUTINE subprogram. However, the following rules are also applicable:

1. An argument may not appear on the left side of an equals sign and must not be redefined in any way.
2. The name of the FUNCTION must appear on the left side of an equals sign at least once in the subprogram. (This is how the result of the calculations within the subprogram is returned to the calling program.)

Example:

```
// JOB      X      Y      Z
// FOR SUBR
  FUNCTION SUBR(X,I,Z)
  DIMENSION Z(10)
  .
  .
  .
  SUBR = X*Z(I) + 10.835
  .
  .
  RETURN
  END
// FOR PROGM
  DIMENSION ARRAY(10)
  .
  .
  XYZ = SUBR(ROLXZ,2,ARRAY) * 4 + 82.18
  .
  ABC = SUBR(ROLAC,8,ARRAY)
  .
  .
  .
  END
// DUP
*STORECI M      1 PROGM PROGM RSTRT
*CCEND
// JOB      X      Y      Z
// END OF ALL JOBS
```

5.2 ASSEMBLER LANGUAGE SUBROUTINES

SUBROUTINE and FUNCTION subprograms to be called by a FORTRAN program may be written in Assembler language. However, a knowledge of the coding generated by the compiler is required. The linkage to each subprogram is generated by the FORTRAN compiler in the same manner as an Assembler language CALL. The argument list consists of a string of DC statements defining the addresses of the arguments.

At core load build time the Assembler CALL is replaced by a BSI instruction.

Example:

```
FORTRAN  CALL  SUBR  (I, J, K)
ASSEMBLER
          CALL  SUBR  (BSI L SUBR)
          DC    I      Address of I
          DC    J      Address of J
          DC    K      Address of K
          .      (Next Executable Instruction)
          :
          :
          I      DC    40
          J      DC    6
          K      DC    18
```

A SUBROUTINE subprogram returns the results of its computations by means of the argument list.

Example:

```
//JOB  X Y Z
//ASM
          ENT      SUBR  Define Entry Point
SUBR      DC      0      Entry Point
          LDX  L I  SUBR  XRI = Argument Address
          LD   L I  0      LD I
          A    L I  1      Add J
          STO  L I  2      Store in K
          BSC  L I  3      Return to Caller
          END
//DUP
*STORE      SUBR
//JOB  X Y Z
//END OF ALL JOBS
```

A FUNCTION subprogram returns its result via the floating accumulator for floating-point results or the A-register for integer results.

Example:

```
FORTRAN CALL K = SUBR (I, J)
//JOB      X Y Z
//ASM
          ENT      SUBR Define Entry Point
SUBR      DC        0      Entry Point
          LDX      LI SUBR XRI = Argument Address
          LD       LI 0      Load Argument 1
          A        LI 1      Add Argument 2
          *
          BSC      LI 2      Return to Caller
          END
//DUP
*STORE      SUBR
//JOB      X Y Z
//END OF ALL JOBS
```

For a subprogram with floating-point results the following coding may be used to get to the floating accumulator (words 41, 42, and 43 on the work level):

```
          :
          :
          LIBF   FLD
          DC     ANSWR
          :
          :
ANSWR     DEC    8.6
```

5.3 REENTRANT CODING

Since TSS is a multilevel operating system, a subroutine may be called from different levels of execution. Care must be taken that partial or intermediate results are not overlaid when subsequent calls are made to the subprogram. This objective is accomplished by assigning to each interrupt level a block of 104 words, 42 words of which are available for storage by user-written programs. Also locations 54 and 55 in the fixed area of core, the contents of the A- and Q-registers, the carry and overflow indicators, and index registers 1, 2, and 3 are saved and subsequently restored by MIC. MIC then sets word 104₁₀ to the base address of the current interrupt level work area. By loading index

register 3 (usual case) indirectly with the contents of word 104 and using this as a base register for indexed instructions, different effective addresses are generated for each work level used. Note that the examples in paragraph 5.2 are reentrant since all data is held in the registers.

An alternative method would be to mask the system and thus prevent interrupts from being recognized. For short sequences this method may be the most efficient one to use. However, for long sequences it is undesirable since it lengthens response time to interrupts.

5.4 TIMER SERVICING SUBROUTINES

There are three hardware timers on the 18/30. Under TSS the user has access to the first two: timer A and timer B. Timer C is used by the system for internal timing and to provide for nine software timers that are very similar – from an operational standpoint – to hardware timers. Thus the user has access to 11 timers.

To access a hardware timer, the following call is made:

```
CALL    TIMER    (name, n, int)
name      symbolic name of the subprogram to be executed when the
           timer goes to zero
n         timer number (timer A = 1, timer B = 2)
int      number of timer intervals to elapse before executing
           subprogram name
```

NOTE: Name must appear in an EXTERNAL statement. Also a subprogram cannot reference itself in a CALL TIMER statement.

The software timers are accessed by using the CALL COUNT statement.

When the System Director is assembled, the user has an option of specifying how many COUNT subprograms will be included at skeleton build time. A

maximum of 32 is allowed. At skeleton build time the user-written subprograms are included in the skeleton by use of a special *INCLD card. It has the following format:

*INCLD name/llbb

name symbolic name of the subprogram to be included in the skeleton
 ll constant (26 or 27) that specifies the group of count routines to which name is assigned (26 for routines 0-15 or 27 for routines 16-31)
 bb specific number (0-15) within the group

Example:

<u>Count Routine No.</u>	<u>ll</u>	<u>bb</u>
0	26	00
1	26	01
2	26	02
:	:	
:	:	
15	26	15
16	27	00
17	27	01
18	27	02
:	:	
:	:	
31	27	15

The user can execute these subprograms by using the CALL COUNT statement which has the format:

CALL COUNT (nt, int, ns)

nt number of the program timer to use (1 through 9)
 int number of timer intervals to elapse before execution
 ns number of the subprogram to execute. This is the number assigned during skeleton build.

Note that in the CALL COUNT statement no subprogram is referred to by name, only by number. Therefore, a COUNT subprogram can reschedule itself for execution by using a CALL COUNT statement.

5.5 CORE LOADS

A core load is a module that resides on disk in an immediately loadable, executable format. It consists of a program and any subroutines, library subprograms, or functions referenced by that program which are not part of the system skeleton. Linkages to skeleton resident subroutines are determined when the core load is built and placed on the disk. Execution of the core load begins with the System Director's loading the core load as if it were data (no relocation or correction of references is necessary). All core loads are loaded into variable core, starting at the beginning address of variable core. When the loading process is complete, the System Director branches to the beginning of the program which was used as the basis for the building of the core load. This address was symbolically specified by the END statement when the program was assembled or was generated automatically by the FORTRAN compiler.

There are three basic types of core loads: nonprocess, process mainline, and interrupt. Nonprocess core loads are executed under the Nonprocess Monitor in either the time-sharing or off-line mode of operation (off-line pack only). This type of core load is meant to perform functions which are not related to the on-line system or functions which are performed infrequently, at the request of the system operator.

Process mainline core loads execute process-related functions as part of the on-line system. The execution of process mainline core loads is determined by a job queue within the skeleton as well as by direct core-load-to-core-load sequencing using skeleton-resident linkage routines.

Interrupt core loads are loaded and executed in response to process interrupts. When the specified interrupt occurs, variable core is saved in the interrupt save area on disk. The interrupt core load is loaded for execution, and all other levels which might be serviced by out-of-core interrupts are masked. These levels remain masked until the interrupt core load has completed its

execution. At the completion of the interrupt core load, variable core is restored so that processing at the mainline level (a process mainline core load or a nonprocess function) can continue. Only one out-of-core interrupt is allowed for each interrupt level. One interrupt core load may not interrupt another interrupt core load regardless of the priority level assignments of the two core loads.

The same core load can be used as an interrupt core load or a process mainline core load by being designated as a combination core load when it is built. This type of core load must be written to conform to the restrictions for both interrupt and process mainline core loads.

5.5.1 Core Load Coding

The bulk of the coding is the same for programs used to construct any type of core load. System considerations such as response time, available core, and available disk space determine the type of core load used to perform a particular function. Each core load type is initiated by a different element of the system. For proper system performance it is necessary that each type exit via the correct system routine. Basic restrictions are imposed on each type core load so that the system can operate efficiently. Coding a program to be used as the basis for a core load requires a familiarity with the restrictions and capabilities of each core load type.

5.5.2 Nonprocess Core Loads

Nonprocess core loads perform functions which are not related to the on-line processor or which are run infrequently at the request of the system programmer or operator. This type of core load is always executed under the Nonprocess Monitor, usually during time-share. Nonprocess functions can be run off-line under TASK only if an off-line disk pack is used.

There are two types of nonprocess core loads. Both types are built and executed as the result of a nonprocess job stream. Type 1 is built in and

executed from the temporary (nonprocess working storage) disk area. Since each //JOB card causes the clearing of the temporary disk area, this type must be built each time it is executed. Type 2 is built in the temporary disk area, but is subsequently stored in the fixed disk core load area, and is executed from the fixed area.

There are more restrictions placed on type 1 than on type 2 nonprocess core loads. Table 5-1 summarizes the capabilities and restrictions on each type of nonprocess core load. The restrictions shown in this table refer to the use of system subroutines only. For example, a nonprocess core load cannot mask the system using the MASK subroutine. It can, however, perform a direct XIO in Assembler language which would accomplish the same results. The restrictions placed on nonprocess core loads are safeguards for on-line operation integrity and should be observed. The restrictions that apply to nonprocess programs refer also to subroutines called by nonprocess programs.

The coding for a nonprocess core load consists of any legal FORTRAN or Assembler language statements. Two exits are provided. The normal method of termination for a nonprocess core load is the CALL EXIT. This call returns control to the Nonprocess Monitor which searches the control deck for the next //JOB card. If a second nonprocess core load has been stored in the fixed core load area (i. e. , a type 2 nonprocess core load), its execution may be initiated by a CALL LINK (NAME). A call to the LINK routine terminates the execution of the present nonprocess core load and initiates the loading for execution of the named nonprocess core load. The named core load must have been stored in the fixed area prior to the execution of the CALL LINK.

If FORTRAN is used, the *NONPROCESS PROGRAM control card should be part of the compilation job stream for the program and any subroutines used by the program. This allows for the checking of system references so that

Table 5-1. Summary of Capabilities and Restrictions of Nonprocess Core Loads

<p>I. <u>Capabilities Type 1 and Type 2</u></p> <ul style="list-style-type: none">● May be written in FORTRAN or Assembler language● May use INSKEL subroutines for I/O● May use system disk files● May use nonprocess working storage for temporary working files● May use any data processing peripheral● May link to another nonprocess core load if that core load resides in the fixed disk area <p>II. <u>Restrictions Type 1 and Type 2</u></p> <ul style="list-style-type: none">● Must be executed under time-sharing● May not use system process I/O subroutines● May not mask the system● May not initiate a programmed interrupt <p>III. <u>Capabilities of Type 2 Not Applicable to Type 1</u></p> <ul style="list-style-type: none">● May alter mainline queue table● May reference and set system clock● May use hardware or programmed timers● May sense mask status of system● May clear recorded interrupts

nonprocess programs do not disrupt the on-line system. Use of the COMMON statement refers to the high core area used by process mainline core loads. Note, however, that no communication between process and nonprocess core loads can be attained through this area, since the functions are completely separate. This area can be used as a communication area between the non-process program and its subroutines or between linked nonprocess core loads. INSKEL COMMON may be referenced by nonprocess core loads used to change system parameters.

Figure 5-1 depicts the job stream required to build and execute a type 1 nonprocess core load. The main program and each subroutine are assembled as individual relocatable modules. These may be stored in the relocatable program area (LET area), but it is not necessary that they be. The relocatable modules are placed in the temporary disk area automatically if the assembly is error free. If the permanent LET area is not used, the modules must be assembled each time the core load is to be built for execution.

The // XEQ card is used to build and execute the core load when used as shown. Columns 16 and 17 of this card must contain blanks indicating that the core load must be built before execution. Columns 8 through 12 contain the name of the relocatable program to be used as the basis for the core load building process. The cards following the // XEQ card are control cards for the core load builder. If FORTRAN disk I/O is used, these cards describe the files that the FORTRAN program references (see summary of Nonprocess Monitor control cards).

Figure 5-2 depicts the job stream required to build and execute a type 2 nonprocess core load. The assembly of each element is identical to a type 1 nonprocess core load. The building sequence contains an important addition. In this sequence the core load is built and stored in the core load area before execution. The core load will remain in this area until deleted by a separate job sequence.

The *STORECI card is used by the core load builder as the source of the basic information necessary to build the core load. An explanation of the contents of this card can be found in the summary of Nonprocess Monitor control cards.

Since the core load is stored in the fixed disk area, later execution can be accomplished using the shorter job stream shown in the figure. The job stream used to delete the core load from the fixed area is shown in figure 5-3.

```

// JOB      X Y Z
// FOR TEST
:
:
CALL SUB1
:
:
CALL SUB2
:
:
END
(FORTRAN Source Program)
// FOR SUB1
SUBROUTINE SUB1
:
:
END
// FOR SUB2
SUBROUTINE SUB2
:
:
END
// XEQ TEST
*CCEND
:
:
(Data Cards if Required by Program)
// JOB      X Y Z
// END OF ALL JOBS

```

Figure 5-1. Build and Execute a Type 1 Nonprocess Core Load

```

// JOB      X Y Z
// FOR TEST
:
CALL SUB1
:
CALL SUB2
:
:                                     (FORTRAN Source Program)
END
// FOR SUB1
SUBROUTINE SUB1
:
END
// FOR SUB2
SUBROUTINE SUB2
:
:
END
*STORECI      TEST  TEST
*CCEND
// JOB      X Y Z
// END OF ALL JOBS
:
// JOB      X Y Z
// XEQ TEST  FX
:
:                                     (Data Cards if Required by Program)
// JOB      X Y Z
// END OF ALL JOBS

```

Figure 5-2. Build and Execute a Type 2 Nonprocess Core Load

```
// JOB      X  Y  Z  
  
// DUP  
  
*DELET          TEST  
  
// JOB      X  Y  Z  
  
// END OF ALL JOBS
```

Figure 5-3. Delete a Type 2 Nonprocess Core Load

Example:

```
//JOB X Y Z
//FOR NPRCS
*NONPROCESS PROGRAM ← (Specifies Nonprocess Program;
                        Mainline Only)
:
:
CALL EXIT ← (Last Executable Statement in
            END                               Nonprocess Program)
//DUP
*STORECI      NPRCS      NPRCS
//JOB X Y Z   ─┬─ (No Restart Core
//END OF ALL JOBS   Load Specified)
                  (No Core Load Type Specified)
```

5.5.3 Process Core Loads

Process core loads perform functions directly related to the process that are not extremely time critical. This might include a control program for a slow reaction process, a logging program, or a data collection program. It is executed by placing its name in the mainline queue table. When it is the highest priority program in the queue, the VIAQ routine will read it into core and execute it.

The only restrictions to process programs are as follows:

1. The last logical statement must be one of these:
 - CALL VIAQ
 - CALL CHAIN (NAME)
 - CALL DPART
2. None of the following statements may be included:
 - CALL ENDTS
 - CALL EXIT
 - CALL INTEX

Example:

```
//JOB X Y Z
//FOR PRCSS
```

:

```
CALL VIAQ
END
```

```
//DUP
```

```
*STORECI M PRCSS PRCSS RSTRT
```

```
//JOB
```

```
//END
```

(If *NONPROCESS PROGRAM is not specified, this is automatically a process program.)

(Last executable statement in process program)

└────────── (Restart Core Load Name)

5.5.4 Interrupt Core Loads

Interrupt core loads are loaded and executed in response to process interrupts. Combination core loads (i. e. , core loads that can be executed as both an interrupt core load and a process mainline core load) must conform to the rules for both types of core loads.

A typical interrupt core load is outlined in the following example.

Example:

```
//JOB X Y Z
//FOR INTCL
```

:

```
CALL INTEX
END
```

```
//DUP
```

```
*STORECI I 1 INTCL INTCL
```

```
//JOB
```

```
//END
```

(Written as Process Program)

(Last Executable Statement in Interrupt Core Load)

LLBB

└────────── (Level and Bit to respond to)

└────────── (No Restart Specified)

└────────── (Specifies Interrupt Core Load)

5.6 INSKEL INTERRUPT SERVICING ROUTINE

```
//JOB      X Y Z
//FOR      SUBROUTINE INTRP
          :
          :
          CALL INTEX
          END
//DUP
*STORE INTRP
//JOB      X Y Z
//FOR      PROGM
          :
          :
          CALL VIAQ
          END
//DUP
*STORECI M 1 PROGM PROGM RSTRT
*INCLD INTRY/0604
*CCEND
//JOB      X Y Z
//END
```

To include this subprogram in the skeleton use the same *INCLD card when building the skeleton:

```
          :
          :
//XEQ SKBLD
*INCLD INTRP/0604
          :
          :
*CCEND
```

5.7 CORE LOAD BUILDING

To build a core load all relocatable modules must be on the disk. However, the modules may be loaded to the nonprocess working storage area of the disk from cards immediately prior to execution of the core load builder. This feature may be used to provide a backup on cards in case of a system failure, to conserve disk room, or to send a compiled or assembled program to another user.

The method outlined below may be used to generate the object decks.

```
//JOB X Y Z
//FOR SUB1
*PUNCH
  SUBROUTINE SUB1
  :
  : (FORTRAN Subroutine)
  :
  RETURN
  END
  :
  : (Blank Cards)
  :
//JOB X Y Z
//FOR PROGM
*PUNCH
  :
  :
  CALL SUB1
  :
  :
  CALL VIAQ
  END
  :
  : (Blank Cards)
  :
//JOB X Y Z
//END OF ALL JOBS
  :
  :
```

Later the object decks generated above can be used to generate a core load:

```
//JOB X Y Z
*STORE T RD 0 SUB1
  :
  : (SUB1 Object Deck)
  :
*STORECI M RD 1 PROGM PROGM RSTRT
  :
  : (PROGM Object Deck)
  :
*CCEND
//JOB X Y Z
//END
```

The occurrence of the //JOB card at the end of the deck has erased any reference to the relocatable modules SUB1 and PROG1 in the nonprocess work storage area. The core load PROG1 is permanently stored on drive 1, and an entry has been made in FLET to that effect.

The core load can be built from the source cards without punching an object deck. Again, the relocatable modules are left in nonprocess working storage and are not placed in the permanent user's area with resulting entries in LET:

```
//JOB X Y Z
//FOR SUB1
  SUBROUTINE SUB1
  :
  :                               (FORTRAN Subroutine)
  :
  RETURN
  END
//FOR PROG1
*NONPROCESS PROGRAM
  :
  :
  CALL SUB1
  :
  :
  CALL EXIT
  END
//XEQ PROG1
*CCEND
//JOB XYZ
//END
```

Note that there is no //JOB card between the subroutine and the main program. A //JOB card reinitializes all references to the temporary (nonprocess work storage) area. Thus, a //JOB card would have erased any reference to SUB1, and the core load builder would not have been able to include it in the core load.

If the modules previously compiled were stored in the LET area as well as being punched into cards, the following sequence would be sufficient to build the core load.

```

//JOB   X Y Z
//DUP
*STORECI M 1  PROG M 1  RSTRT
*CCEND
//JOB
//END

```

5.8 DISK STORAGE AREAS

There are several storage areas on disk with which the user is concerned.

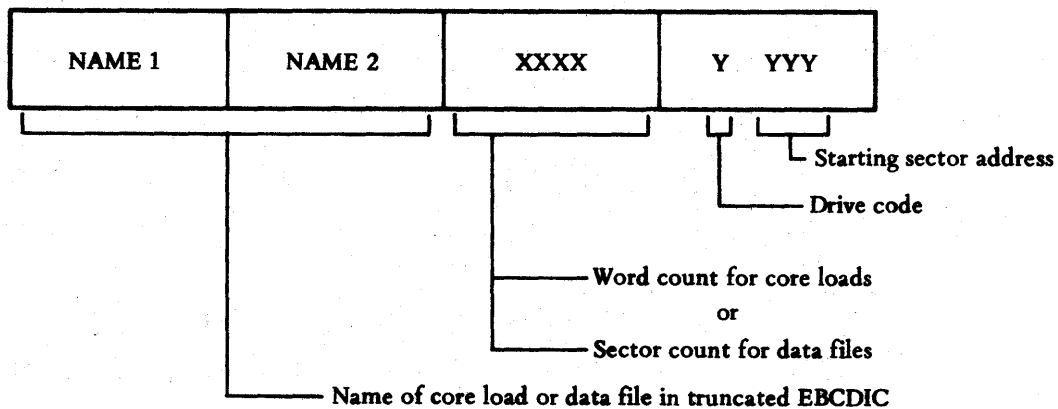
Figure 5-4 shows a disk storage arrangement.

5.8.1 Core Load Area

The core load area is reserved for storage of core image programs and data files. Each program or file in this area is assigned a fixed location and an entry in FLET. Therefore, additional core loads or data files will not overlay those previously defined.

5.8.2 Fixed Location Equivalence Table (FLET)

FLET serves as a disk map for the location of core loads and data files stored in the process core image storage (or core load) area. Each file or core load has an entry in FLET. As the user stores additional core loads or data files on the disk, additional entries are made in FLET. A FLET entry has the following format:



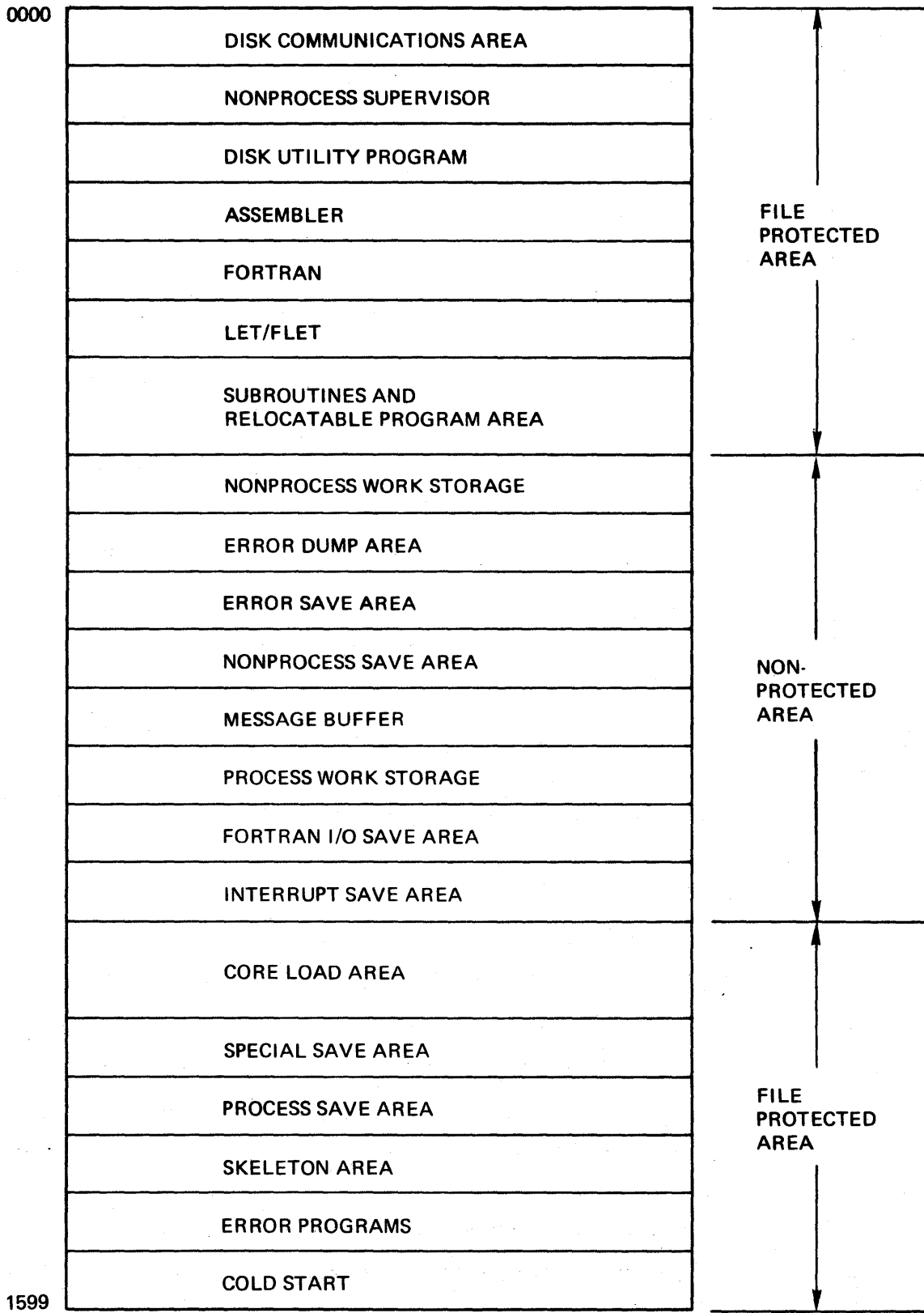


Figure 5-4. Disk Storage Arrangement

There are two ways to make an entry in FLET:

- The *STORECI function of DUP is used to build a core image program and store it in the core load area. The relocatable mainline program used to generate the core load may be in the relocatable area on the disk or it may be in the form of a binary object deck entered through the card reader.
- The *STOREDATA function of DUP is used to store data from cards to the working storage or fixed areas. To reserve an area in the core load area for future use as a data file a *STOREDATA to the fixed area is used.

5.8.3 Working Storage

There are two areas of working storage on the disk that are used as temporary storage by system programs and user-written process or nonprocess core loads. There are no entries in FLET for files set up in this area. Thus, any data left in this area after execution of a system or user-written program may be overlaid by another program. The nonprocess working storage area is used by the Nonprocess Monitor and by user-written nonprocess core loads. The process working storage area is used only by process programs. A process and/or nonprocess work storage area may be assigned to each drive.

5.8.4 FORTRAN Disk Input/Output

To use the disk I/O routines in a FORTRAN program, an *IOCS(DISK) control card must precede the source program at compilation time. This control card can be used only with mainline programs. Subroutines using disk I/O do not use IOCS control cards. However, a mainline program that calls a subroutine which uses disk I/O must use an *IOCS(DISK) control card.

The DEFINE FILE statement is used to communicate to the FORTRAN compiler the quantity and size of the data records within files that are used by a particular program and its associated subroutines. Subroutines use the files as defined by the main program so the DEFINE FILE statement will appear only with a

mainline program. The purpose of the DEFINE FILE statement is to divide the disk area into files to be used in the disk READ, WRITE, and FIND statements.

The format of the DEFINE FILE statement is as follows:

DEFINE FILE i (n, l, U, v)

- i a positive integer constant that is the symbolic designation for this file. The value of i must be less than or equal to 32,767.
- n an integer constant that defines the number of file records in this symbolic file.
- l an integer constant specifying the number of words (length) of each file record in this symbolic file. The value of l must be less than or equal to 320.
- U the letter U designates the file must be read/written with a disk READ/WRITE statement; no data conversion is involved.
- v a nonsubscripted integer variable name that is set at the end of execution of each disk READ, WRITE, and FIND statement referencing this symbolic file. After a READ or WRITE statement v is set to the value of the next available file record. After a FIND statement v is set to the value of the indicated file record. This variable must appear in COMMON if it is referenced by more than one program at execution time.

The format of the READ, WRITE, and FIND statements is as follows:

READ (i'r) list

WRITE (i'r) list

FIND (i'r)

- i the symbolic file number; may be an unsigned integer constant or an integer variable.

- r** an integer expression designating the record number where transmittal will start; may be an integer expression.
- list** a list of variable names, separated by commas, for the input or output data.

The **FIND** statement moves the disk read/write head to the specified record. Its use is not required to perform disk operations.

The ***FILES** control card is used to assign symbolic FORTRAN files to specific areas on the disk. The format of the ***FILES** control card is as follows:

```

1
*FILES (i, NAME, d)           (No embedded blanks are allowed.)
i           the symbolic file number that corresponds to the file number
           in the DEFINE FILE statement in the FORTRAN source
           program.
NAME        the name of the data block to use. This is the same as the
           name established by the *STOREDATA control card. When
           this parameter is used, the third parameter is ignored.
d           the logical drive (0, 1, or 2) to use if a name is not present
           in the second parameter.

```

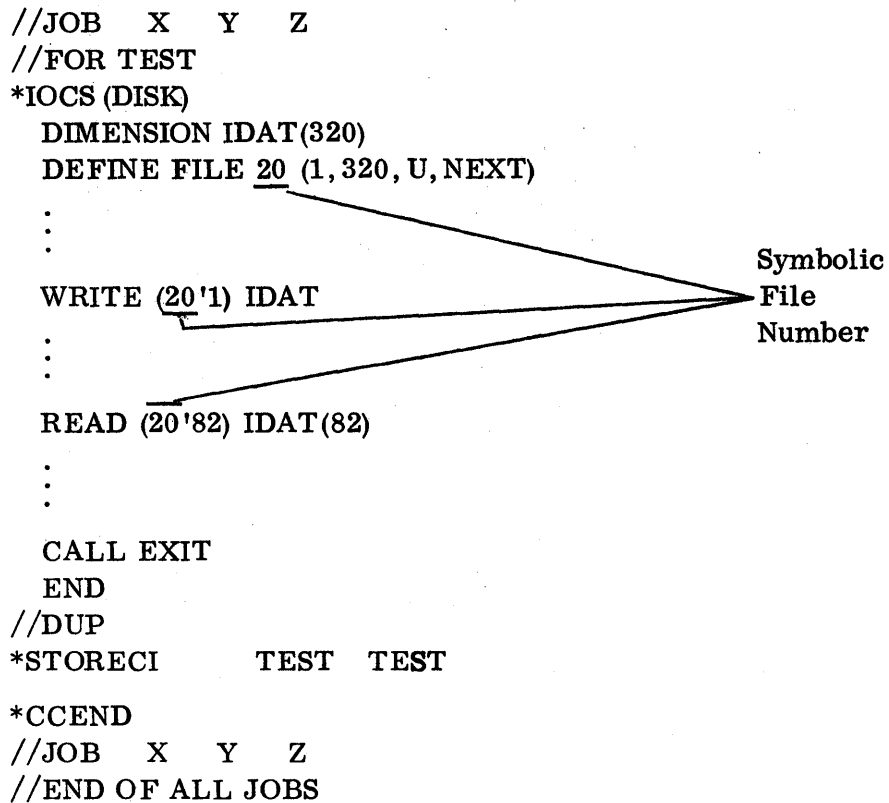
Disk Storage Options. Three options exist for the assignment of files to specific disk areas.

1. Do not include a ***FILES** card for a file that was defined in a FORTRAN source program. The file length specified in the source program is reserved in either process work storage or nonprocess work storage. If more than one file is defined, successive files are set up in the disk work area below the first file. Files that are assigned to disk work storage by the loader for this option will all be located on the temporary disk drive assigned by the **//JOB** control card.

Example:

```
//JOB X Y Z
//FOR TEST
*IOCS (DISK)
  DIMENSION IDAT(320)
  DEFINE FILE 20 (1, 320, U, NEXT)
  :
  WRITE (20'1) IDAT
  :
  READ (20'82) IDAT(82)
  :
  CALL EXIT
  END
//DUP
*STORECI TEST TEST
*CCEND
//JOB X Y Z
//END OF ALL JOBS
```

Symbolic
File
Number



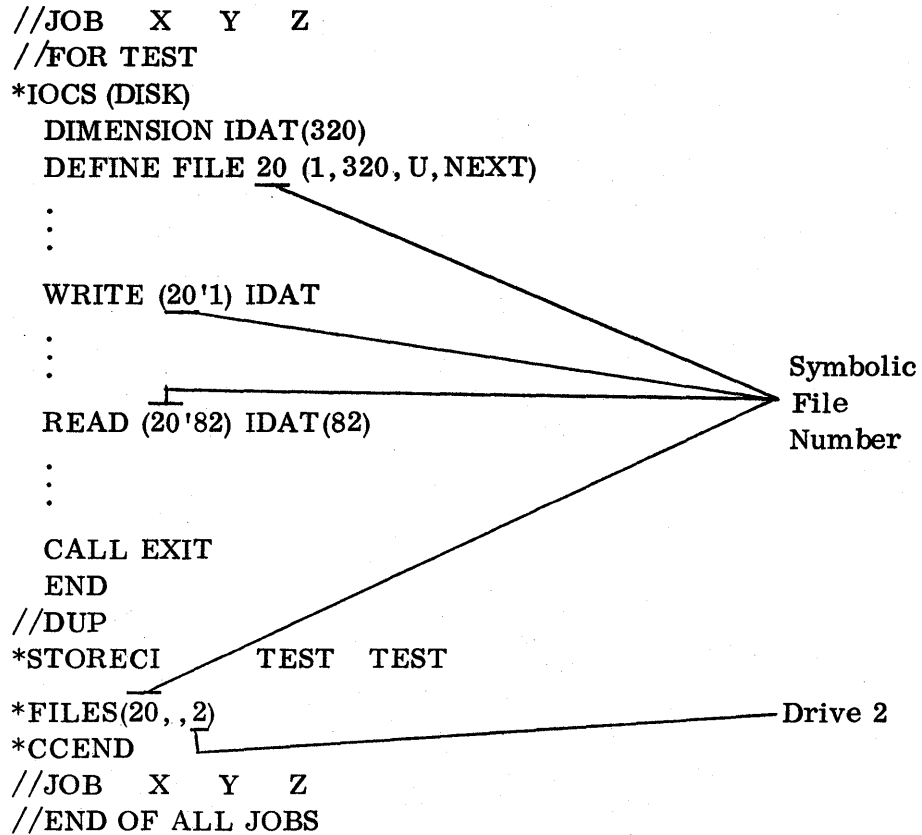
2. Assign the files to work storage on a specific drive. This method of storage is the same as described in item 1, above, but the user must include a *FILES card to assign the drive number. For this option, no name is specified on the *FILES control card.

Example:

```
//JOB X Y Z
//FOR TEST
*IOCS (DISK)
  DIMENSION IDAT(320)
  DEFINE FILE 20 (1,320,U,NEXT)
  :
  :
  WRITE (20'1) IDAT
  :
  :
  READ (20'82) IDAT(82)
  :
  :
  CALL EXIT
  END
//DUP
*STORECI TEST TEST
*FILES(20,,2)
*CCEND
//JOB X Y Z
//END OF ALL JOBS
```

Symbolic
File
Number

Drive 2

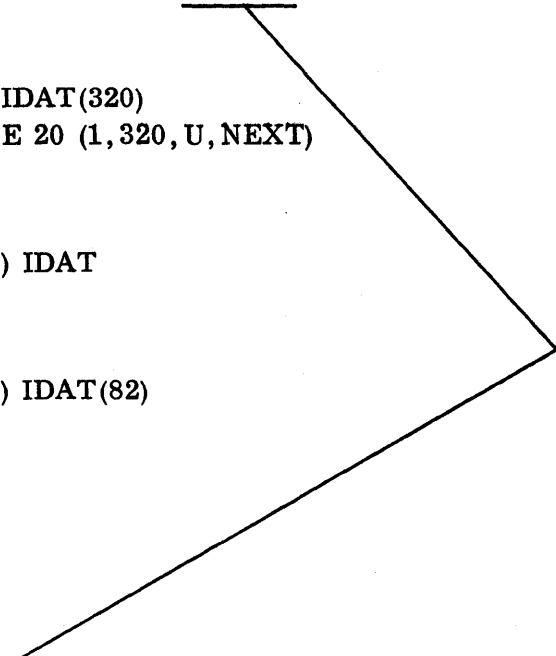


3. Write the files in the permanent areas of the disk. This option requires that the area on disk that is to contain the files must be named in FLET. Including this name as the second parameter of the *FILES statement causes the third parameter (drive selection) to be ignored.

Example:

```
//JOB X Y Z
//DUP
*STOREDATAD WSO FX2 NAMEX 001 00320
//FOR TEST
*IOCS(DISK)
  DIMENSION IDAT(320)
  DEFINE FILE 20 (1,320,U,NEXT)
  :
  :
  WRITE (20'1) IDAT
  :
  :
  READ (20'82) IDAT(82)
  :
  :
  CALL EXIT
  END
//DUP
*STORECI NAMEX
*FILES(20,NAMEX,2)
*CCEND
//JOB X Y Z
//END OF ALL JOBS
```

Symbolic
File
Name



5.8.5 Assembler Disk Input/Output

The DSA statement allows a symbolic reference to a previously defined data block in the core load area on the disk without requiring the specific location of the data file or core load. The format of the DSA statement is as follows:

label	DSA	namex
label		the current value of the Location Assignment Counter when the DSA statement is encountered.
namex		name of a data area previously stored on the disk with an *STORECI or *STOREDATA function.

The Assembler reserves three words in the object program for each DSA statement. These words are filled in by the Core Load Builder.

For a data file these words will contain:

WORD 1	LENGTH (IN SECTORS)
WORD 2	SECTOR ADDRESS
WORD 3	NOT USED

To operate on an entire data file, the length (sector count in word 1) must be multiplied by 320 before a disk call is executed.

For a core load these words contain:

WORD 1	LENGTH (IN WORDS)
WORD 2	SECTOR ADDRESS
WORD 3	NOT USED

Example:

```
//JOB X Y Z
//DUP
*STOREDATAD WSO FX2 NAMEX 001 00320
//ASM TEST
      TEST LLD IOAR
           M D320
           SLT 16
           STO IOAR
           :
           LIBF DISKN
           DC /2000
           DC IOAR
D320 DC 320
           :
           IOAR DSA NAMEX
           BSS 319
           :
           END TEST
//DUP
*STORECI TEST TEST
*CCEND
//JOB X Y Z
//END OF ALL JOBS
```

Symbolic
File
Name

In writing to a file in the work storage areas of the disk, the displacement option in the DISKN control parameter is used. The displacement option is the fourth hexadecimal digit of the control parameter. When it is set to 1, the starting address of either the process or nonprocess work storage area for the specified disk (depending on which type of program called DISKN) is added to the sector address in the I/O area to generate the effective sector address.

Example:

```
//JOB X Y Z
//ASM TEST
      TEST LIBF DISKN
      DC /2001
      DC IOAR
      DC 0
      :
      :
      IOAR DC 320 Word Count
      DC 0 Relative Sector Address
      BSS 320 Data Area
      :
      :
      END TEST

//DUP
*STORECI TEST TEST
*CCEND
//JOB X Y Z
//END OF ALL JOBS
```

5.9 INPUT/OUTPUT SUBROUTINES

Input/output subroutines are provided with the TSS system to relieve the user of the burden of handling all the details peculiar to each device. The provided subroutines are:

<u>Subroutine</u>	<u>Name</u>
Card I/O	CARDN
Disk Storage	DISKN
Printer/Keyboard	TYPEN
Teletypewriter Printer	WRTYN
Printer	PRNTN
Magnetic Tape	MAGT
Paper Tape I/O	PAPTN
Plotter	PLOTX

The CARDN, DISKN, and TYPEN (or PRNTN) subroutines must be included in the skeleton area since they are required by the System Director. The rest of the subroutines may either reside in the skeleton area or be loaded into VCORE as needed.

The disk and printer routines may be referenced from any interrupt level, masked or unmasked. All other I/O subroutines may be called only from interrupt levels that are lower than the interrupt level associated with the device being used. These subroutines assume the necessary interrupt levels are unmasked so that the associated I/O interrupts can be recognized.

Each of the provided I/O subroutines operates as two units:

- The call unit, which is entered when a user's calling sequence is executed.
- The interrupt response unit, which is entered as a result of an I/O interrupt.

Except for DISKN, which does not save the contents of the A-register, all the I/O subroutines save and restore the contents of the A- and Q-registers, the index registers, and the overflow and carry indicators. In addition to saving these register and indicator values, the call unit of each subroutine determines whether any previous operation on the specified device is still in process, verifies the legality of the calling sequence, saves the calling sequence, and initiates the requested I/O operation.

When an interrupt is received during an I/O operation, the interrupt response unit checks for errors, performs any necessary data manipulation, initiates character operations, and -- in case of errors -- initiates retry operations. This unit transfers control to the MIC routine which returns control to the user's program.

The subroutines are referenced through calling sequences, which provide the information (parameters) necessary for the subroutines to operate properly.

The calling sequences for all the I/O subroutines have the same format:

LIBF	subroutine mnemonic
DC	control parameter
DC	I/O area
DC	special condition parameters

A synopsis of the function of each of the I/O subroutines is given in the following paragraphs.

5.9.1 Card I/O Subroutine – CARDN

CARDN performs all input/output functions relative to the card reader and card punch. Although this subroutine is not reentrant for a single device, it may be called from different levels for different devices. CARDN must be located in the Skeleton I/O area.

For a read function CARDN first clears and then stores a -1 in each word of the user-specified I/O area, initiates the input operation to read one card into the input area, and returns control to the user's program. The data will be stored in the input area in the format specified by the user.

For a punch function CARDN punches into one card the number of columns of data specified by the word count at the beginning of the user's I/O area. The character punched is the image of the leftmost 12 bits in the word.

For a test function CARDN determines whether a device is busy. If the device tested is not ready (i.e., the previous operation has not been completed), control is returned for the calling program at location LIBF+2. If the device tested is ready, control is returned to location LIBF+3. When CARDN determines a device is not ready on an I/O operation, it notifies EAC and then returns control to the user's program (at the proper location depending on which function was called). The requested operation is retained by CARDN

and will be executed the next time the routine is called and before the new operation, if the device is ready. If the device is not ready, CARDN waits until it becomes ready before executing both calls. Therefore, the user should have a test function after each read, punch, and feed operation to ensure that -- if the device was not ready at the time of the call for the I/O function -- the function can still be executed when the device does become ready.

For a feed operation CARDN advances all cards in the device to the next station; e. g. , a card at the punch station is moved to the stacker. During a feed operation no data is read or punched.

5.9.2 Disk Storage Subroutine - DISKN

DISKN performs all reading and writing of data from and to the disk. This subroutine must be resident in the Skeleton I/O area and may be called from any interrupt level, masked or unmasked.

DISKN is the only one of the input/output subroutines that does not save and restore the contents of the A-register.

For a test function DISKN may be requested to perform either of two tests:

- Determine if the specified device is not busy
- Determine whether the input/output area referenced is in use by any disk device

Control is returned to location LIBF+3 if the test indicates busy, or to location LIBF+4 if the test indicates not busy.

For a read function DISKN positions the access arm and reads data into the user's input/output area until the specified number of words have been transmitted. Should a read check error occur during reading, DISKN will attempt ten times to re-read the data. If the error still exists, the read function is terminated, and control is transferred to EAC.

For a write with readback check function DISKN determines whether the sector address specified for writing is located in a file-protected area. If it is, DISKN transfers control to EAC. If the specified sector is not file protected, DISKN writes the contents of the I/O area onto disk. Writing a partial sector clears the remainder of the sector to zero. A readback check is performed on the data written. If the check identifies an error, the write operation is repeated and rechecked (the function may be repeated up to ten times). If the error persists, DISKN transfers control to EAC.

For a write without readback check function the operation is similar to that for write with readbackcheck except that the check is not performed.

For a write immediate function DISKN writes data to the specified sector with no attempt to recognize any error. To ensure validity of data written to disk, use of the write with readback check function is recommended. Writing a partial sector clears the remainder of the sector to zero.

For a seek function DISKN may be requested to:

- Seek to the group of sectors specified by the sector address in the user's input/output area.
- Seek to the next group of sectors, regardless of the specified sector address. However, that sector address must be a valid one.

For a seek home function DISKN performs a seek to cylinder zero and ignores the sector address specified in IOA. All interrupts are masked during the operation of this function. Note that for a cold start or reload procedure a seek home function must be the first disk operation called on a given disk.

5.9.3 Printer/Keyboard Subroutine – TYPEN, WRTYN

A single subroutine handles input and output for both the printer/keyboard and the teletypewriter. If the system configuration does not include a printer/keyboard, the keyboard portion of the subroutine is omitted during system

generation. The subroutine may be called by either of two names: TYPEN or WRTYN. At assembly time the user must define two conditions for this subroutine:

- Whether or not messages to the teletypewriter are to be buffered to disk
- The message unit size required for each teletypewriter

This subroutine can control up to eight teletypewriters.

The test function is common to TYPEN and WRTYN. The subroutine determines whether the specified device is busy. If the device tested is busy, control is returned to the calling program at location LIBF+2. If the device tested is ready, control is returned to location LIBF+3.

For a read-print function the subroutine reads from a keyboard and prints on a specified printer the requested number of characters. Three keyboard functions are recognized by the printer/keyboard subroutine:

- Backspace (ER CHR). The backspace key is used to indicate that the previously entered character was in error. The subroutine overprints that character with a slash and then is ready to accept a new character to replace the erroneous one.
- Reentry (ER FLD). The reentry key is used to indicate that an entire message is in error. The subroutine spaces to a new line and then is ready to accept a new message to replace the erroneous one.
- End-of-Message (EOF). The end-of-message key is used to indicate that a message is completed. The subroutine places an NL character in the input/output area and terminates operations. (If the message terminates because the word count reached zero, there will not be an NL character in the I/O area.)

For a print function the subroutine outputs the specified number of characters on the specified printer. The data to be printed must be in typewriter output-code, packed two characters per 16-bit word. Control characters may be embedded in the message. Note that a carriage return to a new line is not automatic when the subroutine is called.

A message priority may be indicated in the calling sequence. When messages are buffered to disk, four message priorities exist:

- EAC messages
- Keyboard entries
- Priority messages
- Normal messages

When messages are not buffered, there are only two priorities:

- EAC messages
- Keyboard, priority, and normal messages

To notify the system that data is to be entered, the operator simultaneously presses the CONTROL and A keys on the keyboard. This action causes the Printer/Keyboard Subroutine to set up an interrupt to a lower level. At system generation time the user must have specified the level for this interrupt and must have provided an interrupt program to service such interrupts.

5.9.4 Printer Subroutine – PRNTN

PRNTN handles all printing and carriage control functions for the line printer. One line of data can be printed or one carriage operation can be performed with each call to this subroutine. The data to be printed must be in the user's I/O area in BCD form, packed two characters per word.

For a test function PRNTN determines whether the previous operation has been completed. If that operation has not been completed, PRNTN returns control to the user's program at location LIBF+2. If the previous operation has been completed, PRNTN transfers control to location LIBF+3.

For a print with checks function PRNTN prints the data from the user's I/O area and checks for print errors. If an error is detected, PRNTN branches to EAC after the line has been printed.

For a print without checks function PRNTN prints the data from the I/O area, but ignores any print errors.

For a carriage control function PRNTN performs the specified carriage operation (e. g. , skip to indicated channel).

For a read carriage control tape function PRNTN determines which control tape channel is on and stores a bit in the A-register to reflect the condition.

5.9.5 Magnetic Tape Subroutine - MAGT

MAGT performs all magnetic tape handling functions: test, read, write, and control. The calling sequence for the MAGT subroutine must specify a special condition routine. This routine will be entered when any of the following conditions occurs: end-of-tape interrupt, end-of-file read, wrong length record read, intermediate read or write check, read or write check on last record.

For a test function MAGT determines whether the previous operation has been completed. If that operation has not been completed, control is returned to the user's program at location LIBF+2. If the previous operation has been completed, control is returned to location LIBF+3.

Read functions may be requested with or without error retries. For a read with error retries operation MAGT reads one record and checks for errors. If an error occurred, the subroutine will re-read the record until the record is read correctly or until 100 retries have been performed. If the record is read successfully, MAGT exits normally. If the error persists after the 100 retries, MAGT branches to EAC, which will return control to the special condition routine. For a read without error retries operation MAGT reads a record, checking only for special conditions. Detection of such a condition causes MAGT to branch to the special condition routine; otherwise, MAGT exits normally.

Write functions may also be requested with or without error retries. The operation for the write functions is the same as for the read operations except that data is output instead of input and the number of retries is 3.

There are six control functions:

- Rewind. MAGT initiates a tape rewind operation and exits normally.
- Rewind and unload. MAGT rewinds the tape to load point and exits normally.
- Backspace. MAGT backspaces the tape one record, unless the tape is already positioned at load point in which case the operation is ignored.
- Write tape mark. MAGT writes a tape mark on the tape.
- Erase. MAGT erases approximately 3.5 inches of tape.
- Reset. MAGT terminates the tape operation in progress and resets all indicators.

5.9.6 Paper Tape I/O Subroutine – PAPTN

The PAPTN subroutine performs all paper tape input and output operations: test, read, and write. When PAPTN is called, it starts the specified device and, as interrupts occur, transfers data to or from the user's I/O area in core. As data is read, it is packed two characters per word into storage. Data to be punched must appear in the I/O area in the same format.

For a test function PAPTN determines whether the previous operation has been completed. If that operation has not been completed, PAPTN returns control to the calling program at location LIBF+2. If the previous operation has been completed, PAPTN transfers control to the calling program at location LIBF+3.

For a read function PAPTN reads data from a paper tape reader into the specified number of words in the I/O area.

For a punch function PAPTN transfers paper tape characters from the I/O area to the paper tape punch. The operation terminates when a stop code is encountered or when the specified number of words have been read.

The user has the option of specifying that checking is to be performed during the read or punch process. If no checking is specified, operation is as described

above, and no check is made to identify delete or stop characters. If checking is specified, it functions with the PTTC/8 code. The PTTC/8 code for DEL is interpreted as a delete code during a read operation; it is not placed in the I/O area and is not included in the count of words read. The PTTC/8 code for NL is interpreted as a stop code during both read and punch operations. A stop code, encountered during reading, is placed in the I/O area and causes the operation to be terminated. A stop code, encountered during punching, is punched in the paper tape and causes the operation to be terminated. When the specified number of words has been transferred, the operation is terminated whether a stop code has been encountered or not.

5.9.7 Plotter Subroutine - PLOTX

PLOTX converts the hexadecimal digits in the control parameter of the calling sequence into a control word and stores it in a buffer (within the PLOTX subroutine). One digit is stored in the buffer at each call to PLOTX. When the plotter is ready to operate, the movement of the recording pen is controlled by the words in the PLOTX buffer. The pen can be raised, lowered, and moved in any direction.

5.10 SUMMARY OF DUP OPERATION

The Disk Utility Program (DUP) is called by the Nonprocess Supervisor in response to a //DUP control card (table 3-1). DUP is also called automatically following the completion of a successful assembly or compilation. The //DUP control card must be followed by one or more DUP control statements (see table 3-2). These statements define the functions to be performed by DUP. The following paragraphs briefly describe the operation of the routines the DUP control statements reference.

5.10.1 DEFINE Routine

This routine provides five options in performing disk operations. A //JOB card must follow each *DEFINE card.

5.10.1.1 Object Core Size

The OCORE option enables the user to specify the size of object core. Its format[†] is

```
      1
     /-----\
    1         9         5
   *DEFINE OCORE xx
```

where

OCORE identifies the option for defining or redefining the core size of the machine that will execute the system being generated.

xx object system core size:

08 = 8192 words

16 = 16385 words

32 = 32768 words

Use of an *DEFINE OCORE card is not required when the System Loader *DEDIT card is used.

5.10.1.2 Number of Disk Drives

The NDISK option is used to specify the number of disk drives on the system.[†]

```
      1
     /-----\
    1         9         5
   *DEFINE NDISK x
```

where

NDISK identifies the option that alters the communications area (DCOM) to allow the user to change the number of disk drives assigned to the system. When loaded initially, DCOM specifies only one disk drive on the system. This specification must precede skeleton build operation.

x number of drives to be assigned to the system.

[†]Two-digit column numbers are stacked; for example, column number 15 appears as $\begin{matrix} 1 \\ 5 \end{matrix}$.

5.10.1.3 Disk Area Configuration

The CONFIG option allows the user to specify the system configuration with respect to the disk areas.[†]

		1	3	3	4	4	5
1	9	5	2	8	4	9	4

*DEFINE CONFIG x ... x LSKEL LINSV LICP LPWS FS

where

CONFIG identifies the option for defining the user's variable disk area. Multiple user-assigned disk areas may be specified on a single card.

x ... x one to nine alphabetic characters specifying the user areas being defined and the disk drives to which they are assigned. If S is used, it must be in column 15; otherwise, the characters may appear in any sequence. The field is terminated by a blank column. The acceptable alphabetic characters are:

S - must appear in column 15 if used; indicates a group of programs and disk areas (cold start program, etc.) that are a basic set required by the system and that are to be file protected. Must be followed by a numeric character specifying a disk drive, or an X and the numeric character. X designates that a special save area is to be included in the group of programs; i. e., an area for saving VCORE when a CALL SPECL statement is executed.

I - an interrupt save area is to be included; must be used when an S or SX code is used. I must be followed by a numeric character that specifies a disk drive.

P - a process work storage area is to be included; must be followed by a numeric character that specifies a disk drive.

[†]Two-digit column numbers are stacked; for example, column number 15 appears as $\frac{1}{5}$.

M - a message buffer area on disk is required; must be followed by a numeric character that specifies a disk drive.

N - process and nonprocess save areas are required. N is required in generating a time-sharing system; must be followed by a numeric character that specifies a disk drive; can only be used when an S or SX is also used.

F - a FORTRAN save area is to be included; must be followed by a numeric character that specifies a disk drive.

DE - a combined error dump and error program save area is required; must be followed by a numeric character that specifies a disk drive.

E - an error program save area without an error dump area is to be included. This parameter is required by EAC; must be followed by a numeric character that specifies a disk drive; both E and DE cannot appear on the same CONFIG card.

C - a core load area is required on the disk; must not be used if an S code is used; must be followed by a numeric character that specifies a disk drive.

LSKEL the estimated number of words in the skeleton that is being built. This number must be an even decimal number, right justified in the field, and equal to the address of the first word of VCORE.

LINSV number of words in the interrupt save area; must be a decimal number, right justified in the field.

LICP number of groups of eight sectors required for the core load area. Must be used if an S, SX, or C is used. Must be a decimal number (may be all zeros), right justified in the field.

LPSW number of groups of eight sectors in the process work storage area; must be used if a P is used. Must be a decimal number, right justified in the field.

FS number of interrupt levels that use FORTRAN I/O.

5.10.1.4 Remove a Processor

The REMOV routine allows the user to remove a processor from the system.

```
┌──────────────────────────┐ 1  
1            9            5            (See footnote page 5-43.)  
*DEFINE REMOV xxx
```

where

REMOV identifies the option to remove the assembler or FORTRAN compiler from the system.

xxx name of the processor to be removed: ASM or FOR.

5.10.1.5 Condense Relocatable Program Area

The PAKDK routine allows the user to condense the relocatable program area on disk. Programs that are identified by a LET entry of 9DUMMY are eliminated during this operation, thus increasing the working storage.

```
┌──────────────────────────┐ 1  
1            9            5            (See footnote page 5-43.)  
*DEFINE PAKDK x
```

where

PAKDK identifies the option to pack programs in the relocatable program area.

x identifies the disk drive on which the packing is to be performed.

5.10.2 DLABL Routine

The disk label routine enables a user to address a disk pack, enter a new label, change an existing label, or establish a LET or FLET area. Before an *DLABL function can be used, the disk pack must be initialized by the TASK Disk Write Addresses routine.

```
┌──────────────────────────┐ 1    1    4  
1            9 2        9    5            (See footnote page 5-43.)  
*DLABL n ppppp zzz comments
```

where

- n specifies the logical disk drive number (0, 1, or 2).
ppppp a five-digit number; identifies the disk pack; $00000 \leq ppppp \leq 32767$.
zzz designates the size of the LET/FLET area in groups of eight sectors; $000 \leq zzz \leq 199_{10}$.

5.10.3 STORE Routine

The Disk Utility Program's store routine allows the user to store relocatable programs in the relocatable program area on disk. All programs loaded by DUP are automatically file protected. Programs may be loaded from cards to the user or temporary disk areas or from user and temporary disk areas to the permanent user's area on disk. By inserting a 9 punch in column 3 of a data card, the user can direct the store routine to ignore the card checksum.

	1	1	1	2	4
1	1	3	9	1	5

*STORE d ss n progn comments (See footnote page 5-43.)

where

- d destination of data being stored:
T = store from cards to temporary user's area.
≠T } = store to permanent area.
blank }
- ss source of program to be stored:
RD = source is cards.
UA } = source is temporary user's area (same interpretation for any character other than RD).
blank }
- n logical drive number (0, 1, or 2) on which program is to be stored. A blank in this column directs DUP to use the lowest numbered drive that has sufficient space for the program.
- progn name to be assigned to the relocatable program being stored.

5.10.4 STOREDATA Routine

The Disk Utility Program's data storage routine enables the user to store data from cards to the working storage or fixed areas of disk or from working storage to the fixed area. The data is unformatted. All programs loaded by DUP are automatically file protected.

```
1 1 1 1 1 2 3 3  
1 1 3 5 7 9 1 0 6 (See footnote page 5-43.)  
*STOREDATA t ss n ii d progn ccc wwwww
```

where

t indicates type of data being loaded:

D = "true" data; i. e. , high-order bits of the FLET entry are zero.

I = interrupt core load.

C = combination core load.

M = mainline core load.

blank = nonprocess core load.

ss source of input:

RD = source is cards and destination is specified by columns 17-18.

≠RD = source is working storage and destination is implicitly the fixed area.

n logical disk drive number of source is working storage.

ii if input is from cards, ii specifies destination:

FX = store to fixed area.

blank = store to working storage.

d specifies destination:

blank = if destination is fixed area, a search for space is made of FLET on all system drives; if destination is working storage, the temporary drive is searched.

number = only the specified drive is searched. Search is as described above.

progn when destination is a fixed area, this parameter must be a name, left justified in the field (columns 21-25). Name should be the same as used on the associated *FILES cards.

ccc	sector count	}	For type D (column 11) either parameter may be used; word count takes precedence if both appear. For types other than D word count must be specified. If LOCALs occur, both sector count and word count must be specified. Maximum word count for a core load is 65536; maximum sector count (for 64K) is 205.
wwwww	word count		

If input is from cards, a *CCEND card must be the last card in the input stack.

5.10.5 STORECI Routine

The core image storage routine is used to cause a core load to be built and stored in the core image area of disk. All programs loaded by DUP are automatically file protected. When input is from cards, only relocatable mainline programs can be entered. By inserting a 9 punch in column 3 of a data card, the user can direct the storage routine to ignore the card checksum.

	1 1	1 2	2	3	3	
1	9	1 3	9 1	7	3	9

(See footnote page 5-43.)

*STORECI m t rr n name1 name2 name3 llbb

where

m specifies a core map is to be printed; blank specifies no map.

t indicates type of core load to be stored:

I = interrupt core load

C = combination core load

M = mainline core load

blank = nonprocess core load

rr location of relocatable mainline to generate the core image program:

RD = system input device

UA	}	= user area
blank		

n logical disk drive number (0, 1, or 2) on which core load will be stored. A blank in this column directs DUP to use the lowest numbered drive that has sufficient space for the core load.

name1 name to be assigned to the core load being loaded; name is placed in FLET.

name2 name of relocatable program to be converted to core image.

name3 name of mainline core load to be used when the core load being loaded performs a cold start. Required for types (column 11) M and C.

llbb level and bit position for the interrupt serviced by the core load; applicable for types (column 11) I and C.

5.10.6 STOREMD Routine

The modify routine enables a user to make changes to existing nonprocess core loads and relocatable programs without having to delete and replace the entire item. All programs loaded by DUP are automatically file protected. By inserting a 9 punch in column 3 of a data card, the user can direct the modify routine to ignore the card checksum.

	1	1	1	1	2	2	3	
1	9	3	5	7	9	1	7	2
*STOREMD	m	ss	n ₁	pp	n ₂	name1	name2	comments

(See footnote page 5-43.)

where

m specifies a core map is to be printed if the program being loaded is a nonprocess core load; blank specifies no map.

ss source of input:

RD = input is from cards.

$\left. \begin{array}{l} \neq \text{RD} \\ \text{blank} \end{array} \right\} = \text{input is from user's area or temporary area.}$

n₁ logical disk drive number where relocatable program is that is to be used to modify old version. Blank specifies to search temporary area first, then user areas.

pp Location of program to be replaced by new version:

 FX = program to be replaced is a nonprocess core load.

 blank = program being replaced is in relocatable format.

n₂ logical disk drive number where program to be replaced is stored.

name1 name to be assigned to the program or core load being stored.

name2 name of the relocatable main program used to create the core
load; required if columns 17-18 contain FX.

5.10.7 DUMP Routine

The dump routine outputs the specified number of words, sectors, or disk blocks from the user's area, the fixed area, or the working storage area to the system I/O device or the line printer. The same routine may be used to dump programs from the user's area or fixed area to the working storage area.

1	1 1 1 1 2	3	4	(See footnote page 5-43.)
*DUMP	3 5 7 9 1	5	5	
	ss n ₁ dd n ₂ progrn	xxxx	comments	

where

ss location of the program that is to be output:

 WS = working storage (blank implies WS)

 UA = user's area

 FX = fixed area

n₁ logical disk drive number on which the program to be dumped
is located.

dd destination of output:

 PN = system I/O device (blank implies PN)

 PR = line printer

 WS = working storage

n₂ logical disk drive for WS output.

progrn name of program or data area to be output.

xxxx number of disk sectors to be output from nonprocess working
storage; the decimal number is right justified in the field
(columns 35-38).

When the output is from the fixed area to cards, an *CCEND control card is punched as the last card.

5.10.8 DUMPDATA Routine

This routine is similar to DUMP except that it is used to move blocks of data instead of programs. The parameters are the same as those for *DUMP card, except that progn identifies a data area rather than a program.

	1	1	1	1	2	3	4	
1	3	5	7	9	1	5	5	(See footnote page 5-43.)
*DUMPDATA	ss	n ₁	dd	n ₂	progn	xxxx	comments	

5.10.9 DUMPLET Routine

This dump routine outputs the contents of the LET and/or FLET areas from the specified disk drives to the line printer.

	1	2	4	
1	1	1	5	(See footnote page 5-43.)
*DUMPLET	a	n	comments	

where

a specifies the area to be output:

L = dump LET.

F = dump FLET.

blank = dump both LET and FLET.

n disk drive number (0, 1, or 2) when only the contents of a single disk is to be output. Blank indicates all drives are to be dumped.

5.10.10 DELET Routine

The delete routine allows the user to remove a named program, core load, or data file from disk or to replace one core load with another core load of the same type.

	1	2	2	3	4	
1	1	1	7	9	5	(See footnote page 5-43.)
*DELET	t	name1	name2	llbb	comments	

where

t indicates type of program to be deleted:

- D = data
- I = interrupt core load
- C = combination core load
- M = mainline core load
- blank = nonprocess core load

Systems programs and systems areas cannot be deleted with the DELET function (see *DEFINE REMOV).

name1 name of program or core load to be deleted.

name2 name of replacement core load (data can not be replaced).

llbb the 2-digit interrupt level and 2-digit bit position within that level, which is associated with the interrupt or combination core load to be deleted.

5.10.11 SEQCH Routine

The sequence change routine allows the user to alter the sequence of existing core load linkages for process or nonprocess core loads and to change linkages to core loads referenced in the subroutines included in the skeleton. Names are limited to five characters and are separated by blanks (except the replacement name which is separated from the first calling core load name by a comma).

1	8	4	1 (Free Form) ...	(See footnote page 5-43.)
*SEQCH curnt newnn, name1 name2SKEL ... namen				

where

curnt name of current (present) core load being called

newnn replacement core load to be substituted for curnt in the specified core loads

name1-
name2 names of core loads that presently call curnt and in which the calls are to be changed to newnn

.SKEL a name given to the skeleton to allow references in the skeleton to curnt to be changed

5.10.12 DICLE Routine

This routine allows a user to insert an interrupt core load entry in the ICL table for each bit position on each level specified.

```
1      8      4      1
*DICLE progn l1 (b1, b2, ... bn) l2 (b1, b2 ...)
```

(See footnote page 5-43.)

where

- progn name of valid interrupt core load identified in FLET.
- l1 (b1 ...) two-digit level number and two-digit bit positions on which the named core load (progn) is to be run. For process interrupts l1 may be 00 through 23, and b1 ... bn must specify the PISW bit position associated with the hardware interrupts. For programmed interrupts l1 must be 24 or 25, and b1 ... bn must be 00 through 13 or 00 through 09, respectively.

The first blank encountered after column 14 terminates the card.

5.10.13 DWRAD Routine

The Disk Utility Program's disk write addresses routine enables the user to write addresses within a specified area on disk. The disk pack must have been initialized by the TASK Disk Write Addresses program before an *DWRAD control record can be honored.

```
1      1  2  2  3  3  4
*DWRAD n  fff lll Z P  comments
```

(See footnote page 5-43.)

where

- n the number (0, 1, or 2) of the drive containing the disk pack where addresses are to be written. This parameter is required.
- fff first address (hexadecimal value) to be written. Must be divisible by 8.
- lll last address (hexadecimal value) to be written.

- Z specifies that the contents of the disk is to be zeroed when the addresses are written. If column 30 contains a blank or any character other than Z, the disk data will be left unchanged.
- P indicated that disk addresses are to be written with file protection. If a P is not present in column 32, the addresses are not file protected.

5.11 COMMON AREAS

There are two areas of common storage available to the programmer using TSS: the core load COMMON and the INSKEL COMMON. Core load COMMON is used to communicate between a mainline program and its subroutines without passing lengthy argument lists (see paragraph 5.1). It is located at the high end of VCORE and is overlaid by subsequent programs. INSKEL COMMON is used for communication between core loads. It is located in the skeleton between the I/O subroutines and the System Director.

Common areas may be used with both FORTRAN and Assembler language programs. The following FORTRAN statement

```
COMMON I, J, K
```

is equivalent to these Assembler language statements:

```
I    EQU    /FFFF    (-1)
J    EQU    /FFFE    (-2)
K    EQU    /FFFD    (-3)
```

INSKEL COMMON is referenced as follows:

```
COMMON/INSKEL I, J, K
```

The Assembler language equivalent is the same as in the COMMON statement except the base address of INSKEL COMMON must be added to the addresses of I, J, and K. This address is always located in word 156 of the fixed area.

I	EQU		/FFFF	
J	EQU		/FFFE	
K	EQU		/FFFD	
	:			
	:			
	LDX	I1	156	Load Base of INSKEL COMMON
	:			
	:			
	LD	1	I	
	A	1	J	
	STO	1	K	
	:			
	:			
	END			

SECTION 6 – OPERATING CONSIDERATION

This section summarizes frequently required operating procedures. These procedures require that an operating TASK program or a Skeleton Executive be available. (The procedures for system generation, including assembling TASK and building the skeleton, are presented in section 7.)

6.1 OPERATING TASK OFF-LINE

TASK may be operated off-line as a complete operating system. In this case only data-processing activities are carried on; no process control is allowed. When operating in this mode, all functions of the Nonprocess Monitor are available to the user; process core loads may not be built or executed. The programmed interrupts and hardware timers are not available to the user. If any error occurs, the TASK error messages are printed rather than those generated by EAC.

There are two ways to execute a nonprocess program:

- To execute a nonprocess core load previously stored on the disk with the *STORECI function of DUP.
- To build a core load and execute it immediately with no entry made in FLET.

The control card for executing a nonprocess program has the following format:[†]

```
┌────────────────── 1
| 1      8      4
| //XEQ name L
```

or

```
┌────────────────── 1 1
| 1      8      6 8
| //XEQ name FXn
```

[†]Two-digit column numbers are stacked; for example, column number 14 appears as $\frac{1}{4}$.

- name (columns 8-12) is the symbolic name of the relocatable main program to use as a basis for execution or the name of a core load already built and stored on the disk.
- L (optional - column 14) when building a core load, a map of core will be produced if an L is present in column 14. This parameter is ignored if the program is already stored on disk as a core load.
- FX (columns 16, 17) used when the program to be executed already exists on the disk as a core load.
- n (column 18) specifies on which drive the nonprocess core load is located. If this column is blank, all drives beginning with drive 0 will be searched to find the core load.

6.2 TASK DISK WRITE ADDRESSES

The TASK Disk Write Addresses routine writes addresses on a specified disk. It verifies each sector of the disk by reading and writing three different bit patterns. The number of times this process occurs is specified by the user. When an error is encountered during a sector read/write, that sector is rechecked 49 times. Should a second error occur on the same sector, the entire group of eight sectors containing that sector is considered defective. A table of defective sector groups is written on the first sector of the disk. The words written are the logical sector addresses of the first sectors of the defective group of eight sectors. The table is written in words 4, 5, and 6 of sector 0.

The following procedures should be used to load and execute the TASK Disk Write Addresses programs.

1. Load TASK into core.
2. Place the TASK Disk Write Addresses program, followed by two blank cards, in the card reader hopper. Ready the card reader.
3. Set data switches 0 and 15 on to select the TASK absolute loader function.

4. Press STEP switch. After the routine has been loaded, the following message is printed:

TSS DISK WRITE ADDRESSES PROGRAM.

ENTER NO. TRIES ON DATA SW MAX001F.

5. Enter the number of times that the three patterns are to be written on each sector, right justified, in the data entry switches. The number must be a hexadecimal value within the range /0001 to /001F. Press STEP switch.

6. If the number entered is not acceptable, the following message is printed:

ENTER NO. TRIES ON DATA SW MAX001F.

Correct the entry (see step 5). Press STEP switch.

7. If the number of tries entered is acceptable, this message is printed:

DATA SWITCHES EQUAL LOGICAL DRIVE.

DRIVE CODES -- HEX 0000 THRU 0009.

8. Set data switches to the logical drive number of the disk to be initialized, right justified (i. e., /000X). Press console STEP switch.

9. If the number entered is not acceptable, the following message is printed:

ENTER NO. TRIES ON DATA SW MAX001F.

Correct the entry and return to step 5.

10. If the first group of eight sectors (zero) is defective, the disk pack cannot be used by TSS and the following messages are printed:

THIS DISK PACK IS NOT ACCEPTABLE TO TSS BECAUSE OF
EITHER TOO MANY BAD CYLINDERS OR CYLINDER ZERO
IS BAD.

CYLINDERS 0000 ARE DEFECTIVE DO NOT USE SKEL. BLD
WITH THIS PACK.

DATA SW 0 ON GO TO TASK OFF REDO.

Set data switch 0 on to return to TASK or off to return to step 4.

11. If all sector groups are acceptable, the following messages are printed:

THERE ARE NO DEFECTIVE CYLINDERS.

DATA SW 0 ON GO TO TASK OFF REDO.

Set data switch 0 on to return to TASK or off to return to step 4.

12. If there is one defective sector group, the following messages are printed:

CYLINDERS 00AA ARE DEFECTIVE.

DATA SW 0 ON GO TO TASK OFF REDO.

Set data switch 0 on to return to TASK or off to return to step 4. The value 00AA is the logical sector-group number of the defective group of eight sectors.

13. If there are two or three defective sector groups, the disk pack may not be used for the skeleton build function. The following messages are printed:

CYLINDERS 00AA 00AA 00AA ARE DEFECTIVE.

DO NOT USE SKELETON BUILD WITH THIS PACK.

DATA SW 0 ON GO TO TASK OFF REDO.

Set data switch 0 on to return to TASK or off to return to step 4.

14. If there are more than three defective sector groups, the disk pack may not be used by TSS. The following messages are printed:

THIS DISK PACK IS NOT ACCEPTABLE TO TSS BECAUSE OF EITHER TOO MANY BAD CYLINDERS OR CYLINDER ZERO IS BAD.

CYLINDERS 00AA 00AA 00AA 00AA ARE DEFECTIVE.

DO NOT USE SKEL. BLD WITH THIS PACK.

DATA SW 0 ON GO TO TASK OFF REDO.

Set data switch 0 on to return to TASK or off to return to step 4.

15. If there is a seek failure (and the system is unable to recover), the job is aborted; the following messages are printed:

CAN NOT COMPLETE SEEK - ABORT JOB.

DATA SW 0 ON GO TO TASK OFF REDO.

Set data switch 0 on to return to TASK or off to return to step 4.

NOTE: The logical sector-group numbers (N) of the defective groups of eight sectors are printed in hexadecimal notation. To calculate the physical cylinder number (C) of a defective cylinder, use the formula

$$C = N + D$$

where D is the number of defective sector groups preceding the group whose physical number is to be calculated. The sector-group numbers range from 0_{10} to 202_{10} , i. e., from 0_{16} to CA_{16} .

6.3 TASK DISK DUPLICATION PROGRAM

The TASK Disk Duplication program copies the entire contents of one logical disk drive (platter) onto another. All data words, including file protect status, are duplicated. Before this routine is executed, disk addresses must be present on both drives (platters). To load and execute the TASK Disk Duplication program, use the following procedure:

1. Load TASK into core.
2. Place the TASK Disk Duplication program deck in the card reader hopper.
3. Set data switches 0 and 15 on to select the TASK absolute loader function.
4. Press STEP switch. After the routine has been loaded, the following messages are printed:

TSS DISK DUPLICATION PROGRAM.

DATA SWITCHES EQUAL LOGICAL SOURCE DRIVE.

DRIVE CODES -- HEX 0000 THRU 0009.

5. Enter the logical drive number of the drive to be copied from, right justified, in the data switches (i. e., /000X). Press STEP switch.
6. If the number entered is not acceptable, the message shown in step 4 is repeated. Return to step 5.
7. If the number entered is acceptable, the following messages are printed:

DATA SWITCHES EQUAL LOGICAL OBJECT DRIVE.

DRIVE CODES -- HEX 0000 THRU 0009.

8. Enter the logical drive number of the drive to be copied to, right justified, in the data switches (i. e., /000Y). Press STEP switch.

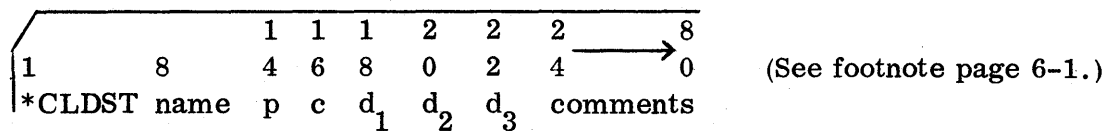
9. If the number entered is not acceptable, the message shown in step 7 is repeated. Return to step 8.
10. If the number entered is acceptable, the following messages are printed:
 COPYING FROM DRV 000X TO DRV 000Y.
 DATA SW 0 ON CONTINUE OFF REDO.
11. If the correct drives are indicated, set data switch 0 on (down), and press STEP switch. If incorrect drives are indicated, leave data switch 0 off, and press STEP switch; the program will print the messages shown in step 4. Continue with step 5.
12. After program execution is completed, the following message is printed:
 DATA SW 0 ON GO TO TASK OFF REDO.
 Set data switch 0 on to return to TASK or off to return to step 4.

6.4 SYSTEM COLD START

A system initialization (or cold start) of the TSS skeleton is accomplished by loading and executing a three-card program using the initial program load (IPL) feature of the GA 18/30 Computer.

6.4.1 Cold Start Name Card

The cold start program is supplied in a ready-to-use format; however, the name card must be punched by the user. Its format is



*CLDST must be punched in columns 1 through 6.

name identifies the first core load to be called. This entry is required. The name is entered in the field left justified, may consist of up to five characters, and must begin with an alphabetic character.

p storage protection option:

blank = no storage protection

1 = storage protection selected

c clock option:

blank = no option selected

1 = clock option selected

d_1, d_2, d_3 logical disk drive assignment ($d_1 \neq d_2 \neq d_3$):

d_1 = required; assigns a physical drive number to logical drive 0. It must be punched 0, 1, or 2.

d_2 = optional; assigns a physical drive number to logical drive 1. It may be blank or punched 0, 1, or 2.

d_3 = optional; assigns a physical drive number to logical drive 2. It may be blank or punched 0, 1, or 2.

comments any comments may be entered in columns 24 through 80.

Columns 7, 13, 15, 17, 19, 21, and 23 must be blank.

6.4.2 Cold Start Procedure

The operating procedure to load and execute the cold start program is as follows:

1. Clear core (see paragraph 6.5).
2. Place the three-card cold start program, followed by the user-punched cold start name card and two blank cards, in the card reader hopper.
3. Ready the reader.
4. Place the system disk pack on drive 0.
5. Ready the disk drive.
6. Set RUN/IDLE switch to IDLE.
7. Set all data switches on the console to the off position (up).
8. Set register select switches 8 and 4 on (down).
9. Press ENTER switch.
10. Reset register select switches (all up).
11. Press ENTER switch.
12. Verify that the HALT switch is up.

13. Press IPL switch.
14. Set RUN/IDLE switch to RUN.
15. Press STEP switch.
16. Follow the instructions printed by the system printer.

6.5 CLEARING CORE

To use properly the storage protect feature of the GA 18/30 Computer, TSS requires that all storage protect bits be set off and that all core locations be set to zero when the system is initialized (cold started or loaded from cards). To accomplish this purpose, a one-card program is loaded (using the initial program load (IPL) feature) into the computer and executed. This program is known as ZAP.

To load and execute ZAP, use the following procedure:

1. Place the ZAP card in the card reader hopper.
2. Ready reader.
3. Set RUN/IDLE switch on console to IDLE.
4. Set all data switches on the console to the off position (up).
5. Set the storage protect override switch (SPO) on (down).
6. Set register select switches 8 and 4 on (down).
7. Press ENTER switch.
8. Reset register select switches (all up).
9. Press ENTER switch.
10. Verify that the HALT switch is up.
11. Press IPL switch.
12. Set RUN/IDLE switch to RUN.
13. Press STEP switch.

When the WAIT light comes on, core is clear.

SECTION 7 - SYSTEM GENERATION

This section gives step-by-step procedures for generating a TSS system from supplied object and source card decks and from subroutines written especially for the system application. Before using these procedures, the reader should be familiar with section 4 of this manual, "System Evolvment."

The minimum configuration for the TSS system is a GA 18/30 Industrial Supervisory System with 8192 words of core memory, a disk drive, a card punch, and a card reader.

7.1 SUMMARY OF SYSTEM GENERATION PROCEDURES

Generating an on-line or off-line system begins with a set of procedures that is common to both types of systems. In this stage a "starter" system called SYSGEN TASK starts the system generation process and directs the writing of addresses on the disk, loading the supplied nonprocess system, assembling TASK and the System Director, and defining the disk system configuration.

In the second stage the procedures depend on whether an on-line or an off-line system is being generated. For an on-line system, skeleton subroutines are compiled, a system skeleton is built, process core loads are built, and an off-line start is performed. For an off-line system relocatable programs are stored on disk from cards, a nonprocess monitor pack is built, and an off-line cold start is performed. A flowchart of these procedures is shown in figure 7-1.

7.2 SYSTEM GENERATION COMPONENTS

A set of supplied card decks and user-prepared control cards are required for system generation.

7.2.1 Supplied TSS System

System object decks, control cards for the standard TSS system, and two source decks are supplied for the system. The system object decks comprise a set of

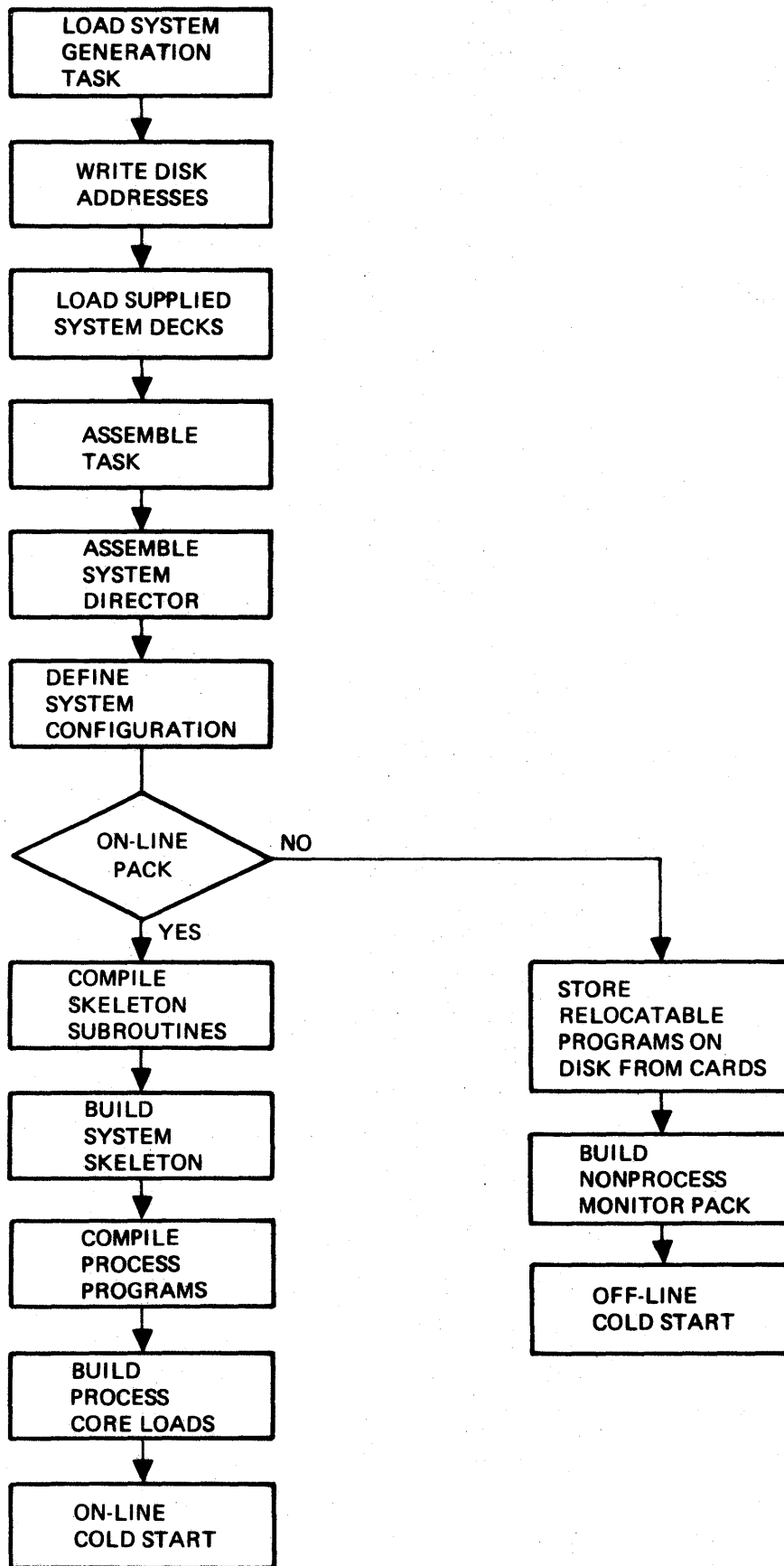


Figure 7-1. System Generation Flowchart

cold start cards, a four-card TASK high-core loader, a SYSGEN TASK deck, a system loader, disc LET/FLET tables, disk communications area, a bootstrap program, a supervisor, a core load builder, a cold start program, a set of disk utility programs, an assembler, a FORTRAN compiler, a subroutine library, a skeleton builder, and stand-alone utilities. The supplied source programs are a TASK and System Director source deck. A set of EQUATE cards for the standard TSS system are included.

7.2.2 User-Prepared Control Cards

The user must prepare control cards such as *ASSIGNMENT and *INCLD cards to tailor the system to the requirements of his process. Any EQU cards that are needed for deviation from the standard system must also be prepared.

7.3 SYSTEM GENERATION PROCEDURES

This section contains the detailed procedures for building an on-line process and nonprocess system as well as an off-line system. Each procedure is given in the form of a table that includes page references to IBM manual descriptions and paragraph references to discussions in this manual. Error codes for any errors that may occur during the procedure are presented in appendix A. The manual references are from IBM 1800 Time-Sharing Executive System: Operating Procedures, Form C26-3754. The procedures start with the object deck shown in figure 7-2.

7.3.1 Loading TASK and Writing Disk Addresses

To generate a TSS system, the user must first load the TASK program and write addresses on the disk. Table 7-1 lists the steps for loading TASK in core. The table is applicable for loading SYSGEN TASK at the beginning of system generation or for loading the operating TASK object deck after TASK has been assembled.

A TASK Disk Write Address object routine is included as part of the utility package supplied with TSS system decks. This routine is loaded and executed by the TASK absolute loader function. The Disk Write Address program writes

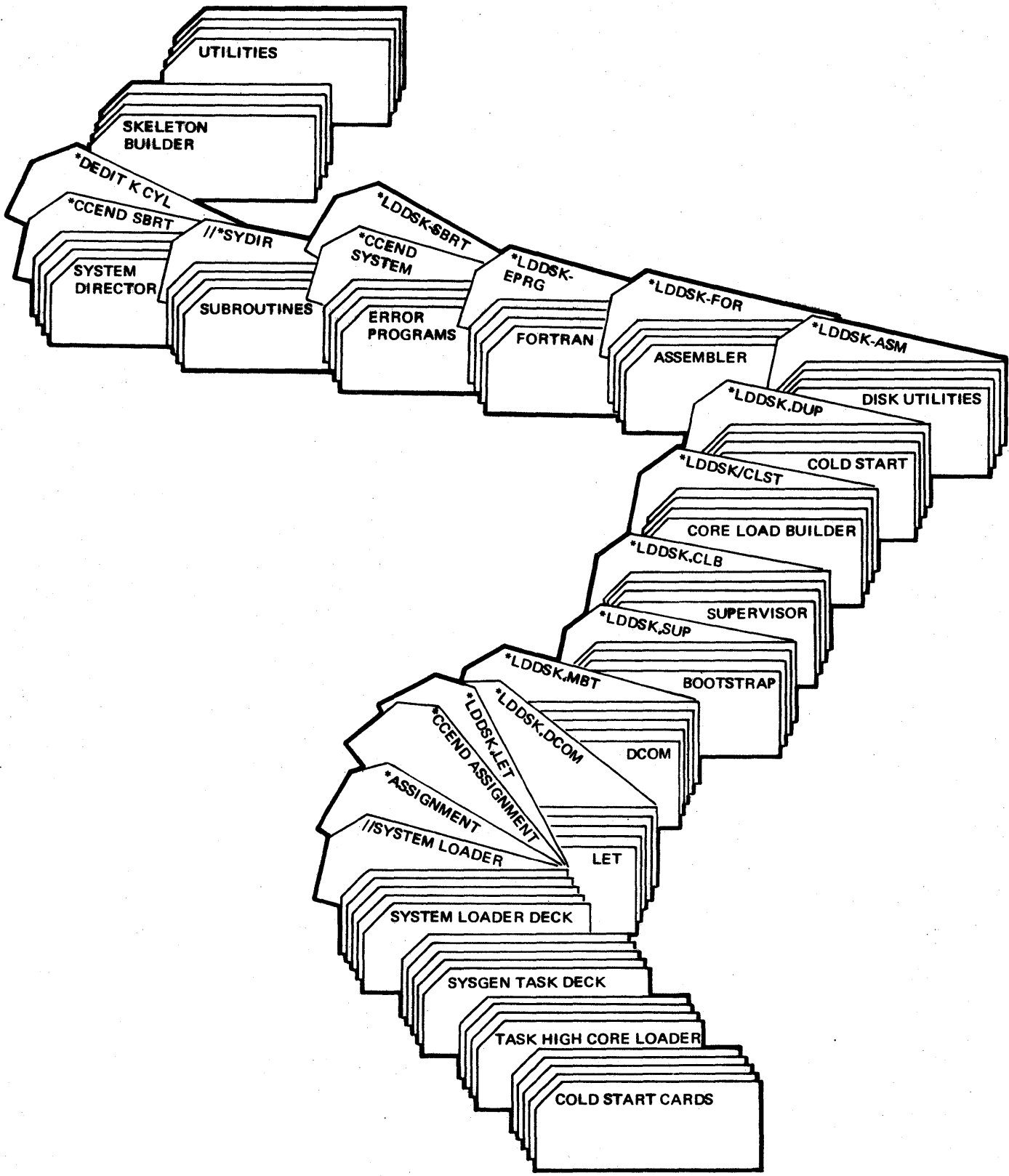


Figure 7-2. Supplied System Object Decks

Table 7-1. Loading TASK in Core

Procedure Step	Page Reference	Paragraph
<ol style="list-style-type: none"> 1. Load the user pack on drive 0 and ready the disk drive 2. Set the RUN-IDLE switch to IDLE (first step in zeroing core) 3. Set the HALT switch on 4. Unlock the WSPS switch by turning it clockwise 5. Set the SPO switch on 6. Load the ZAP card, supplied with the system decks (see figure 7-3) 7. Ready the reader 8. Set all data switches off 9. Set REGISTER SELECT switches 4 and 8 on 10. Press RESET 11. Press ENTER 12. Set the HALT switch off 13. Press the IPL switch 14. Set the HALT switch on 15. Set REGISTER SELECT switches 4 and 8 off 16. Press RESET 17. Press ENTER 18. Set the RUN-IDLE switch to RUN 		

Table 7-1. Loading TASK in Core (Cont.)

Procedure Step	Page Reference	Paragraph
<p>19. Press the STEP switch.</p> <p>20. Set the SPO switch off (last step in zeroing core)</p> <p>21. Place in the card reader hopper the SYSGEN TASK deck without the TASK high core loader or an operating TASK object deck preceded by the four-card high core loader (see figure 7-4)</p> <p>22. Ready the reader</p> <p>23. Select the loading address in the data switches (first step in program load)</p> <p>24. Set the RUN-IDLE switch to IDLE</p> <p>25. Set the HALT switch on</p> <p>26. Set REGISTER SELECT switches 4 and 8 on</p> <p>27. Press RESET</p> <p>28. Press ENTER</p> <p>29. Set the HALT switch off</p> <p>30. Press the IPL switch</p> <p>31. Set REGISTER SELECT switches 4 and 8 off</p> <p>32. Set the HALT switch on</p> <p>33. Press RESET</p> <p>34. Press ENTER</p>	<p>15, 82</p>	<p>4.1</p>

Table 7-1. Loading TASK in Core (Cont.)

Procedure Step	Page Reference	Paragraph
<p>35. Set the RUN-IDLE switch to RUN</p> <p>36. Press the STEP switch (last step in program load). Wait for the following message to be printed:</p> <p style="padding-left: 40px;">GENERAL AUTOMATION 18/30 TSS V3M5</p> <p>DATA SW 0 ON FOR ABSOLUTE LOADER</p> <p>DATA SW 1 ON FOR NONPROCESS MONITOR</p> <p>DATA SW 2 ON FOR SKELETON BUILDER</p>	15	

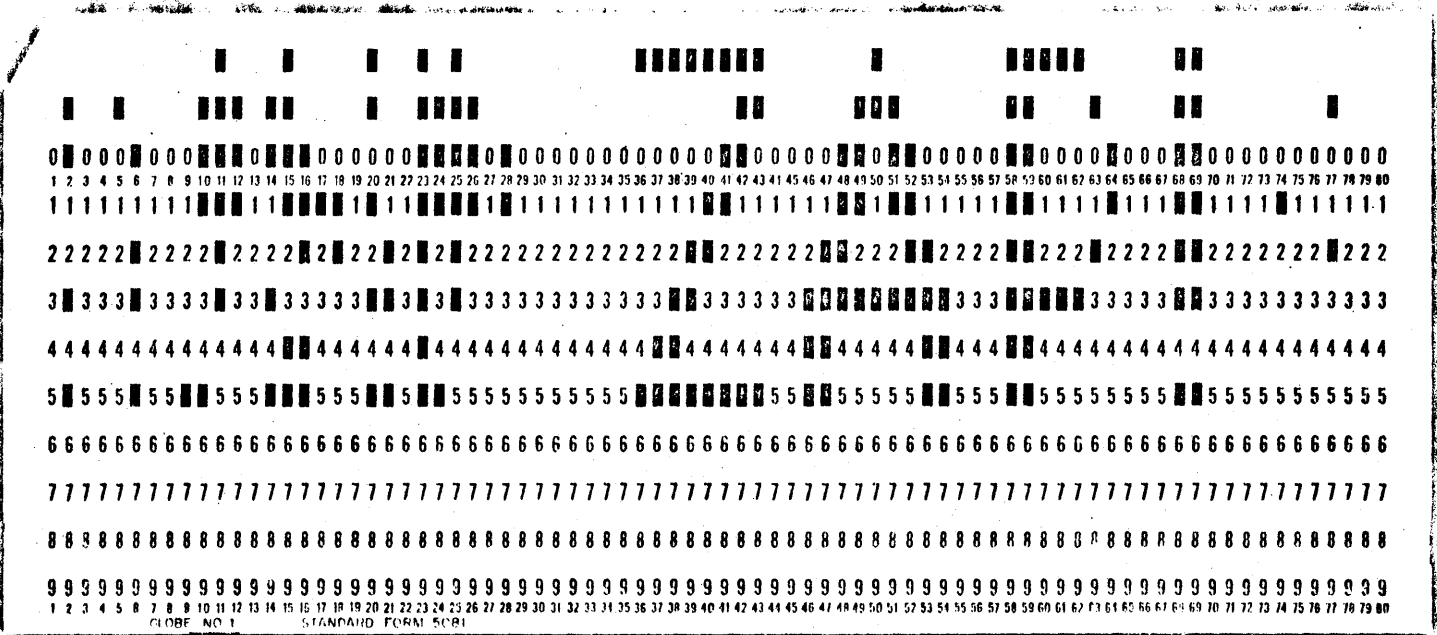


Figure 7-3. ZAP Card

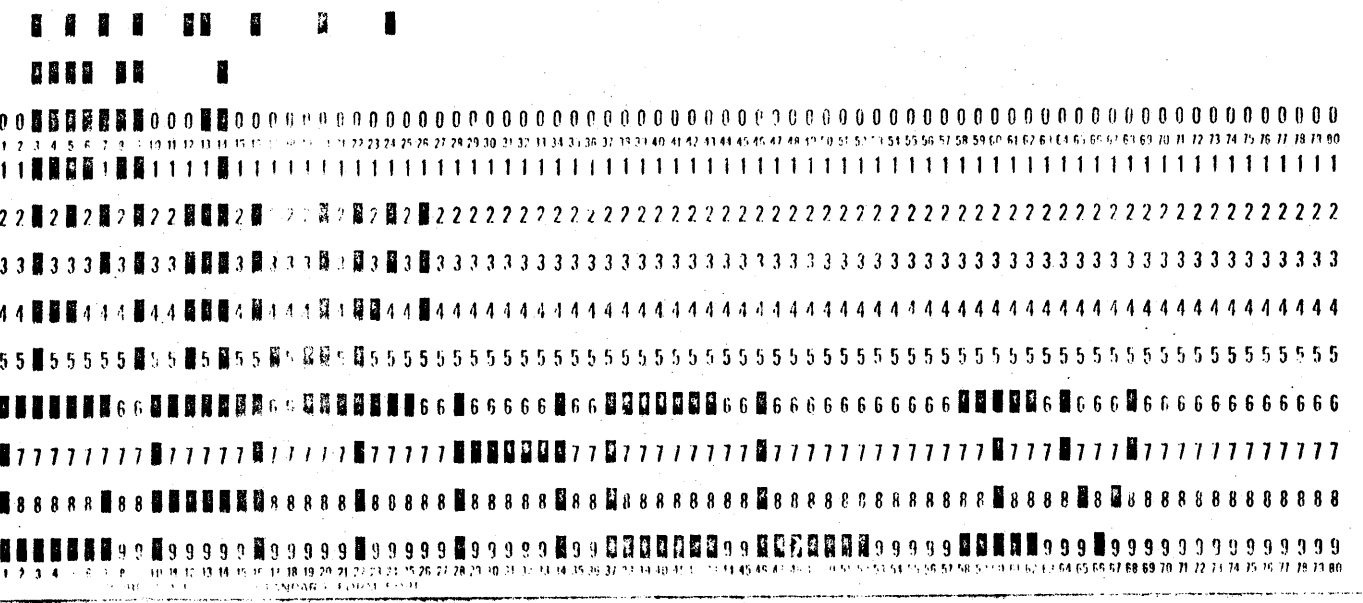
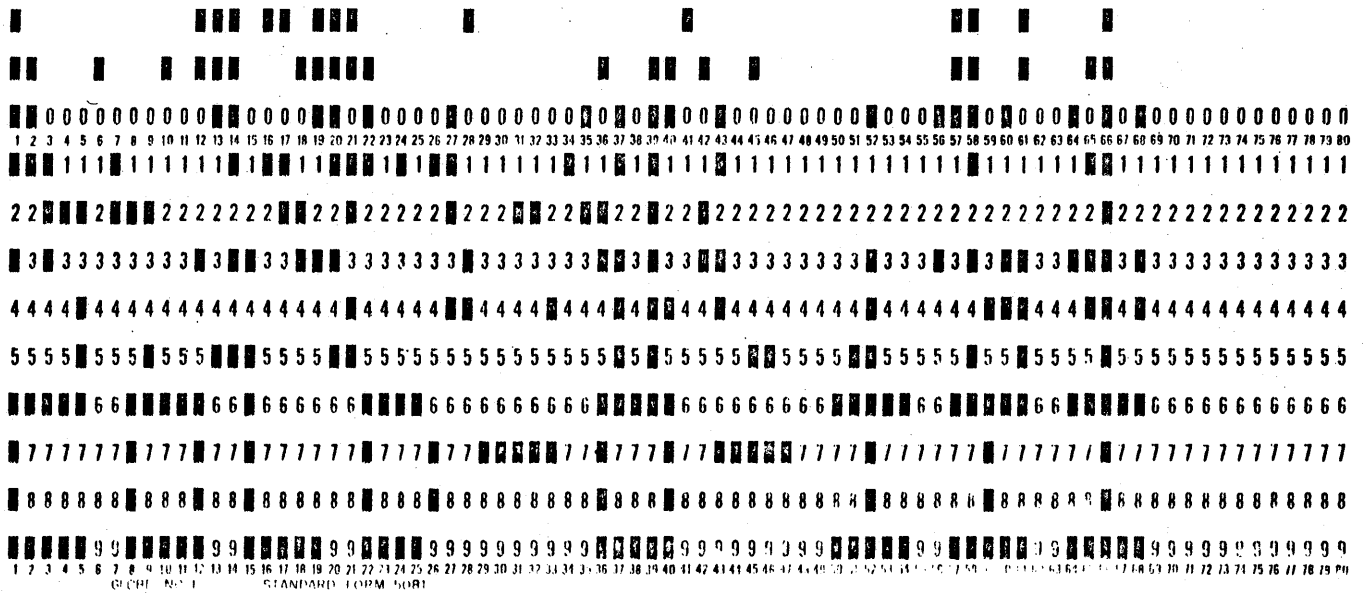


Figure 7-4. TASK High Core Loader Cards (2 of 2)

addresses on a specified disk and checks each sector of the disk by reading and writing three different bit patterns a specified number of times. If an error occurs during a sector read or write, that sector is rechecked 49 times. If a second error occurs on the same sector, the group of sectors containing that sector is considered defective. The logical sector address of the first sector of the defective group is recorded in words 4, 5, and 6 of the first sector of the disk.

To write addresses on the disk, follow the procedure given in table 7-2.

7.3.2 Loading the Supplied System Decks on Disk

To load the supplied system decks on the disk, perform the steps listed in table 7-3.

7.3.3 Assembling TASK

This procedure is presented with the assumption that the system decks have been stored on drive 0 or that a nonprocess monitor pack is available, and that SYSGEN TASK or an operating TASK has been loaded in core. To store the system decks on drive 0, see paragraph 7.3.2. To build a nonprocess monitor pack, see paragraph 7.3.12. To load TASK in core, see paragraph 7.3.1. If the nonprocess monitor pack is being used, perform the cold start procedure described in paragraph 7.3.13.

If the requirements have been met, assemble TASK by performing the steps outlined in table 7-4.

7.3.4 Assembling the System Director

A preassembled standard System Director deck is supplied with the TSS system. If it is necessary to change this program for any reason – for example, to change the base on the time clocks – the supplied source deck should be used and assembled according to the procedure given in table 7-5. It is assumed in this procedure that the system decks have been stored on drive 0 or that a nonprocess

Table 7-2. Writing Disk Addresses

Procedure Step	Page Reference	Paragraph
<p>1. Place in the card reader hopper the TASK Disk Write Address program followed by a blank card</p>	26	4.4, 6.2
<p>2. Ready the card reader</p>		
<p>3. Set data switch 0 on</p>		
<p>4. Set data switch 15 on</p>		
<p>5. Press the console STEP switch. The TASK absolute loader function will be executed. When the routine is loaded, observe the following message:</p>	22	6.2
<p>TASK DISK WRITE ADDRESSES PROGRAM ENTER NO. TRIES ON DATA SW MAX001F</p>		
<p>6. Enter right justified in the data switches the number of times that the three patterns are to be written on each sector. The range of numbers accepted is hexadecimal 0001 to 001F</p>		
<p>7. Press the console STEP switch. If the number entered is not acceptable, the following message is printed:</p>		
<p>ENTER NO. TRIES ON DATA SW MAX001F</p>		
<p>Correct the entry and return to step 6. If the number of tries entered is acceptable, the following message is printed:</p>		
<p>DATA SWITCHES EQUAL LOGICAL DRIVE DRIVE CODES--HEX 0000 0001 0002</p>		

Table 7-2. Writing Disk Addresses (Cont.)

Procedure Step	Page Reference	Paragraph
<p>8. Enter right justified in the data switches the logical drive number (000X) of the disk to be initialized</p> <p>9. Press the console STEP switch. If the 0 group of sectors is defective, the disk pack cannot be used by TSS and the following messages are printed:</p> <p style="padding-left: 40px;">THIS DISK PACK IS NOT ACCEPTABLE TO TSS BECAUSE OF EITHER TOO MANY BAD CYLINDERS OR CYLINDER 0 IS BAD</p> <p style="padding-left: 40px;">CYLINDERS 0000[†] ARE DEFECTIVE</p> <p style="padding-left: 40px;">DO NOT USE SKEL. BLD WITH THIS PACK DATA SW 0 ON GO TO TASK OFF REDO</p> <p>Set data switch 0 on to return to TASK or off to return to step 5</p>		
<p>[†]The logical number L of the defective group of eight sectors (i.e., "cylinder") is printed in hexadecimal notation. To calculate the physical group number P of a defective group of sectors, use the following formula:</p> $P = L + N$ <p>where N is the number of defective groups preceding the group whose physical group number is to be calculated. The group numbers range from 0_{10} to 202_{10}, that is from 0_{16} to CA_{16}.</p> <p>The TASK Disk Write Address program writes a table of defective groups of sectors on the first sector of the disk. The words written are the logical sector addresses of the first sectors of the defective groups of sectors. The table is written in words 5, 6, and 7 of sector 0.</p>		

Table 7-2. Writing Disk Addresses (Cont.)

Procedure Step	Page Reference	Paragraph
<p>If a group of sectors (cylinder) other than group 0 is defective, the following messages are printed:</p> <p style="padding-left: 40px;">CYLINDERS 00XX ARE DEFECTIVE</p> <p style="padding-left: 40px;">DATA SW 0 ON GO TO TASK OFF REDO</p> <p>where 00XX is the logical group number of the defective cylinder. Set data switch 0 on to return to TASK or off to return to step 5. If there are two or three defective groups of sectors, the disk pack may not be used for the skeleton build function. The following messages are printed:</p> <p style="padding-left: 40px;">CYLINDERS 00XX 00XX 00XX ARE DEFECTIVE</p> <p style="padding-left: 40px;">DO NOT USE SKELETON BUILD WITH THIS PACK</p> <p style="padding-left: 40px;">DATA SW 0 ON GO TO TASK OFF REDO</p> <p>Set data switch 0 on to return to TASK or off to return to step 5. If there are more than three defective cylinders, the disk pack may not be used by TSS. The following messages are printed:</p> <p style="padding-left: 40px;">THIS DISK PACK IS NOT ACCEPTABLE TO TSS BECAUSE OF EITHER TOO MANY BAD CYLINDERS OR CYLINDER 0 IS BAD</p> <p style="padding-left: 40px;">CYLINDERS 00XX 00XX 00XX ARE DEFECTIVE</p> <p style="padding-left: 40px;">DO NOT USE SKEL BLD WITH THIS PACK</p> <p style="padding-left: 40px;">DATA SW 0 ON GO TO TASK OFF REDO</p> <p>Set data switch 0 on to return to TASK or off to return to step 5.</p>		

Table 7-2. Writing Disk Addresses (Cont.)

Procedure Step	Page Reference	Paragraph
<p>If all groups of sectors are acceptable, the following messages are printed:</p> <p style="padding-left: 40px;">THERE ARE NO DEFECTIVE CYLINDERS</p> <p style="padding-left: 40px;">DATA SW 0 ON GO TO TASK OFF REDO</p> <p>Set data switch 0 on to return to TASK or off to return to step 5.</p> <p>If there is a seek failure and the system is unable to recover, the job is aborted and the following messages are printed:</p> <p style="padding-left: 40px;">CAN NOT COMPLETE SEEK - ABORT JOB</p> <p style="padding-left: 40px;">DATA SW 0 ON GO TO TASK OFF REDO</p> <p>Set data switch 0 on to return to TASK or off to return to step 5.</p>		

Table 7-3. Loading the Supplied System Decks

Procedure Step	Page Reference	Paragraph
<p>1. Remove from the supplied system object decks the following cards:</p> <ul style="list-style-type: none"> a. Cold start cards b. Skeleton builder c. Disc utility program d. Stand-alone utilities <p>2. If nonreentrant arithmetic, functional, and conversion subroutines are to be used, remove the reentrant versions from the supplied subroutine library and insert the nonreentrant subroutines in their place. The subroutine library can comprise a mixture of reentrant and nonreentrant subroutines, but in no case should both versions of the same subroutine, that is, two subroutines with the same name, be placed in the system. This error will cause the system loader or DUP to generate an error message.</p> <p>In general, the reentrant version of a subroutine should be used if the subroutine is to be called from different levels or is to be included in the skeleton. The reentrant version must be used if an interrupt routine is to be included in a mainline core load, and an interrupt routine on the higher level can interrupt a routine on the lower level during execution.</p>	<p>42</p> <p>39</p> <p>51</p> <p>19</p>	<p>4.9, 6.4</p> <p>4.6</p> <p>5.10</p> <p>6.1</p>

Table 7-3. Loading the Supplied System Decks (Cont.)

Procedure Step	Page Reference	Paragraph
<p style="text-align: center;">Note</p> <p>If the above system requirements are violated, the resulting errors are not diagnosable and the results are unpredictable.</p> <p>The nonreentrant versions of the subroutines may be placed in the subroutine library after the supplied system has been loaded and the skeleton has been built (see table 7-12).</p> <p>3. Insert the system loader assignment cards (IAC and LUN), comments cards, and the *DEDIT card in the supplied system deck as shown in figure 7-5. Assignment cards for the standard TSS system are supplied with the system decks</p> <p>4. Place in the card reader hopper the system decks without the TASK high core loader or the SYSGEN TASK deck, as shown in figure 7-5</p> <p>5. Set data switch 0 on</p> <p>6. Set data switch 15 on</p> <p>7. Press the STEP switch. If any error messages occur during system load, refer to appendix A.</p>	30-32	4.5.6

Table 7-3. Loading the Supplied System Decks (Cont.)

Procedure Step	Page Reference	Paragraph
<p style="text-align: center;">Note</p> <p>If data switch 15 is not on, the following message is printed:</p> <p style="text-align: center;">DATA SW 0 ON LD DISK OFF EXECUTE</p> <p>In this case, ensure that data switch 0 is off and press the console STEP switch.</p> <p>8. Observe the following messages when loading is completed:</p> <p style="text-align: center;">THE SOURCE CORE SIZE IS nnnnnn THE OBJECT CORE SIZE IS nnnnnn DATA SW 0 ON FOR ABSOLUTE LOADER DATA SW 1 ON FOR NONPROCESS MONITOR DATA SW 2 ON FOR SKELETON BUILDER</p> <p>9. Set all console switches to the off position</p>		

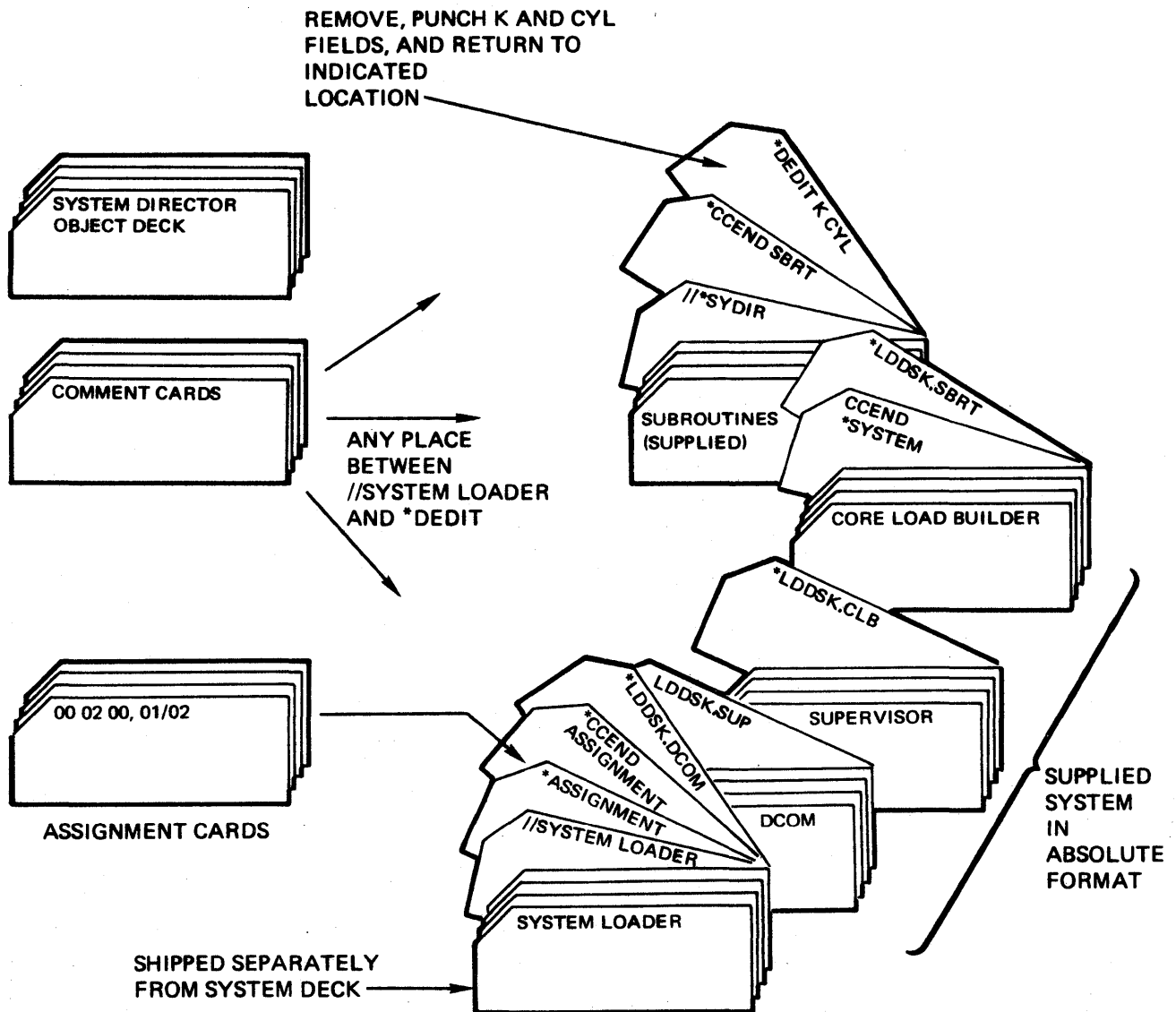


Figure 7-5. Sequence of Control Cards and Systems Decks for TSS System Load

Table 7-4. Assembling TASK

Procedure Step	Page Reference	Paragraph
<p>1. Insert the TASK EQU cards defining the source machine in the TASK source deck as shown in figure 7-6. The EQU cards for the standard TSS system are supplied with the TASK source deck. If a deviation is to be made, EQU cards must be punched. If the same parameter is entered in two or more EQU cards, the entries must be the same</p>	15	4.10
<p>2. If the system has more than one drive, place an entry in the label field of the //JOB card for each drive being used</p>	43	3.2.1
<p>3. Insert the TASK source deck, including the required EQU cards and control cards (figure 7-6) in the card reader hopper. It is suggested that the *LIST card be removed until TASK is assembled without errors</p>	65	3.2.4
<p>4. Ready the card reader</p>		
<p>5. Set data switch 7 on</p>		
<p>6. Press the CONSOLE INTERRUPT switch on the computer console</p>		
<p>7. If assembler errors occur, correct the errors (see appendix A), reload SYSGEN TASK, and return to step 3</p>		

Table 7-4. Assembling TASK (Cont.)

Procedure Step	Page Reference	Paragraph
<p style="text-align: center;">Note</p> <p>The stand-alone card assembler can be used to assemble TASK, although this program is not standard with the TSS system. Before using the 18/30 card assembler, remove the first six cards and the last two cards from the TASK source deck. If any errors occur, repeat the assembling process.</p>		

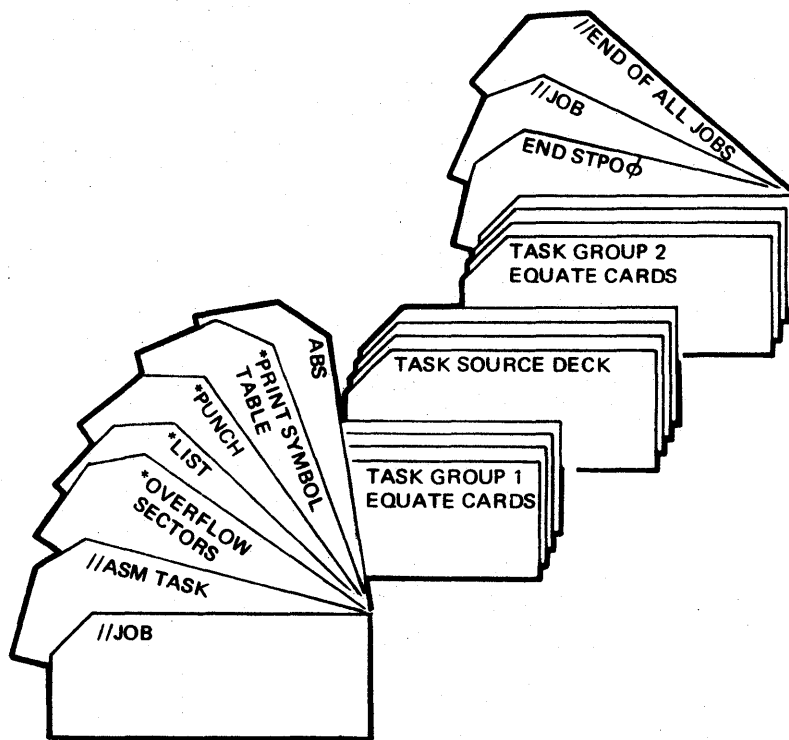


Figure 7-6. TASK Source Deck and Equate Cards

Table 7-5. Assembling the System Director

Procedure Step	Page Reference	Paragraph
<p>1. Insert the System Director EQU cards in the System Director source deck as shown in figure 7-7. The EQU cards for the standard TSS system are supplied with the System Director source deck. If a deviation is to be made, EQU cards must be punched. If the same parameter is entered in two or more EQU cards, the entries must be the same</p>	34	4.11
<p>2. Remove the //DUP and *STORE SYDIR cards from the System Director source deck unless the System Director is to be stored on the pack on drive 0</p>	44	5.10.3
<p>3. If TASK is already in core, proceed to step 4. If TASK is not in core, load TASK as directed in table 7-1 and proceed to step 12</p>	15	4.1
<p>4. Set the HALT switch on (first step in restarting TASK)</p>		
<p>5. Set the RUN-IDLE switch to IDLE</p>		
<p>6. Set the HALT switch off</p>		
<p>7. Enter 0 in the data switches</p>		
<p>8. Ensure that the switches in the bottom row on the console, beginning with the REGISTER SELECT switches, are in the upper position</p>		
<p>9. Press the ENTER switch</p>		

Table 7-5. Assembling the System Director (Cont.)

Procedure Step	Page Reference	Paragraph
<p>10. Set the RUN-IDLE switch to RUN</p> <p>11. Press the STEP switch. Wait for the data switch messages to be printed (table 7-1)</p> <p>12. Insert the System Director source deck in the card reader hopper. It is suggested that the *LIST card following the //ASM SYDIR card be removed until the System Director is assembled without errors</p> <p>13. Ready the reader</p> <p>14. Set data switch 7 on</p> <p>15. Press the CONSOLE INTERRUPT switch on the computer console. If there are no assembler errors, proceed to step 16. If assembler errors have occurred, return to step 4</p> <p>16. Insert the System Director object deck in the system deck (figure 7-5)</p>	<p>65</p>	<p>3.2.4</p>

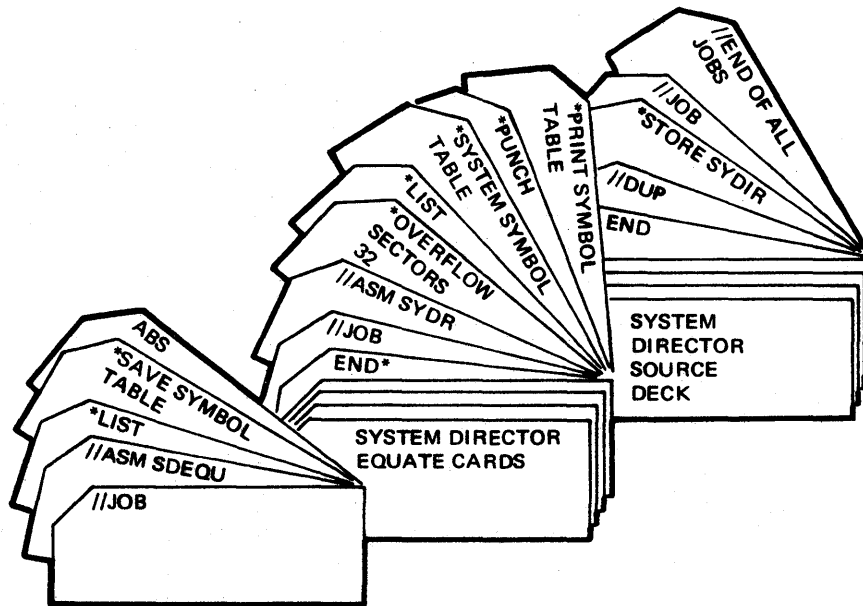


Figure 7-7. System Director Source Deck and Equate Cards

monitor pack is available. To store the system decks, see paragraph 7.3.2. To build a nonprocess monitor pack, see paragraph 7.3.12.

To assemble the System Director, perform the steps listed in table 7-5. The procedure is applicable for initial system generation, when TASK is already in core, and for System Director assembly when TASK has not been loaded.

7.3.5 Defining the System Configuration

It is assumed in this procedure that an initial system generation is being performed or that a new machine configuration is being defined. This is the last procedure in the first stage of system generation, that is, after the system configuration has been defined, the next procedure to be performed is determined by whether the system is to be on-line or off-line. If an on-line pack is desired, proceed to paragraph 7.3.6; if an off-line pack is needed, proceed to paragraph 7.3.11.

To define the system configuration, perform the steps listed in table 7-6.

7.3.6 Compiling Skeleton Subroutines

It is assumed in this procedure that the system decks have been stored on drive 0. To store the system decks, refer to paragraph 7.3.2.

To compile user-written skeleton subroutines, perform the steps listed in table 7-7. The procedure is applicable for initial system generation, when TASK is already in core, and for compiling skeleton subroutines when TASK is not in core.

7.3.7 Building the Skeleton

It is assumed in this procedure that the system decks have been stored, the define configuration function has been performed, the System Director has been stored, user-written skeleton subroutines have been stored, and an operating TASK is in core. To store the system decks, see paragraph 7.3.2. To define

Table 7-6. Defining the System Configuration

Procedure Step	Page Reference	Paragraph
<p>1. Punch *DEFINE CONFIG card</p> <p style="text-align: center;">Note</p> <p>The value of LSKEL and LICP must be calculated by the user. If the user wishes to define his complete system during initial system generation, TASK and the System Director should be assembled before the system configuration is defined.</p>	51, 53	
<p>2. Place the following user-punched cards in the card reader hopper:</p> <pre>//JOB //DUP *DEFINE CONFIG SOM010 ... LSKEL XXXXX LICP XXXX XX //JOB //DUP *DUMPLET //JOB //END OF ALL JOBS</pre>	<p>43</p> <p>44, 51</p> <p>53</p> <p>58</p> <p>45</p>	<p>3.2.1</p> <p>3.2.1</p> <p>5.10.1</p> <p>5.10.9</p> <p>3.2.1</p>
<p style="text-align: center;">Note</p> <p>If the system contains two or more drives, the //JOB card must contain all labels for each nonsystem pack. The *DEFINE CONFIG card can then be used to assign the system areas to the desired drives, for example SOM112.</p>		

Table 7-6. Defining the System Configuration (Cont.)

Procedure Step	Page Reference	Paragraph
<p>3. Ready the reader</p> <p>4. Set data switch 7 on</p> <p>5. Press the CONSOLE INTERRUPT switch on the computer console</p> <p>6. If any error messages occur, correct the errors according to appendix A</p> <p>7. After the //END OF ALL JOBS card is printed, verify the configuration defined by checking the LET/FLET dump printout</p> <p>8. If the configuration is incorrect, punch a corrected *DEFINE CONFIG card and return to step 2</p>	<p>45, 122</p> <p>53</p>	<p>3.2.1</p> <p>5.10.1</p>

Table 7-7. Compiling Skeleton Subroutines

Procedure Step	Page Reference	Paragraph
<p>1. If the user subroutines are not to be stored on the pack on drive 0, remove any *STORE cards from the user-written subroutine source decks</p>	55	5.10.3
<p>2. If TASK is already in core, proceed to step 3. If TASK is not in core, load TASK as directed in table 7-1 and proceed to step 11</p>	15	4.1
<p>3. Set the HALT switch on (first step in restarting TASK)</p>		
<p>4. Set the RUN-IDLE switch to IDLE</p>		
<p>5. Set the HALT switch off</p>		
<p>6. Enter 0 in the data switches</p>		
<p>7. Ensure that the switches in the bottom row on the console, beginning with the REGISTER SELECT switches, are in the upper position</p>		
<p>8. Press the ENTER switch</p>		
<p>9. Set the RUN-IDLE switch to RUN</p>		
<p>10. Press the STEP switch. Wait for the data switch messages to be printed (table 7-1)</p>		
<p>11. Insert the subroutine source decks, with control cards, in the card reader hopper</p>		
<p>12. Ready the reader</p>		
<p>13. Set data switch 7 on</p>		
<p>14. Press the CONSOLE INTERRUPT switch on the computer console. If assembler or compiler errors have occurred, return to step 3</p>		

the system configuration, see paragraph 7.3.5. To assemble the System Director, see paragraph 7.3.4. To store the user-written subroutines, see paragraph 7.3.6. To load an operating TASK, see table 7-1.

To build the system skeleton, perform the steps listed in table 7-8.

7.3.8 Compiling Process Programs

It is assumed in this procedure that the system skeleton has been built and that an operating TASK has been loaded in core. If a skeleton has not been built, refer to paragraph 7.3.7. If TASK is not in core, see table 7-1.

To compile the user's process programs, perform the steps listed in table 7-9.

7.3.9 Building Process Core Loads

It is assumed in this procedure that the user's process programs have been assembled or compiled and loaded to disk and that TASK has been loaded in core. If process programs have not been compiled, see paragraph 7.3.8. If TASK is not in core, see table 7-1.

To build process core loads, perform the steps listed in table 7-10.

7.3.10 On-Line Cold Start

It is assumed in this procedure that process core loads have been built. If the core loads have not been built, see paragraph 7.3.9.

To execute an on-line cold start, perform the steps listed in table 7-11.

7.3.11 Storing Relocatable Programs on Disk from Cards

It is assumed in this procedure that an operating TASK has been loaded in core. If TASK has not been loaded, refer to table 7-1.

If an off-line disk pack is being built, store the relocatable programs on the disk from cards as directed in table 7-12.

Table 7-8. Building the Skeleton

Procedure Step	Page Reference	Paragraph
1. Punch the Skeleton Builder *INCLD cards	46	4.7
2. Insert the skeleton building *INCLD cards in the Skeleton Builder deck as shown in figure 7-8	39	4.7
3. Place on drive 0 the pack that is to become the system pack		
4. Place //JOB and //END OF ALL JOBS cards in the card reader hopper. If the system is being defined for multiple drives, place the proper entries in the //JOB card label fields	43, 45	3.2.1
5. Ready the reader		
6. Set data switch 7 on		
7. Press the CONSOLE INTERRUPT switch on the computer console		
8. Set the RUN-IDLE switch to IDLE (first step in zeroing core)		
9. Set the HALT switch on		
10. Unlock the WSPS switch by turning it clockwise		
11. Set the SPO switch on		
12. Load the ZAP card, supplied with the system decks (see figure 7-3)		
13. Ready the reader		
14. Set all data switches off		

Table 7-8. Building the Skeleton (Cont.)

Procedure Step	Page Reference	Paragraph
<p>15. Set REGISTER SELECT switches 4 and 8 on</p> <p>16. Press RESET</p> <p>17. Press ENTER</p> <p>18. Set the HALT switch off</p> <p>19. Press the IPL switch</p> <p>20. Set the HALT switch on</p> <p>21. Set REGISTER SELECT switches 4 and 8 off</p> <p>22. Press RESET</p> <p>23. Press ENTER</p> <p>24. Set the RUN-IDLE switch to RUN</p> <p>25. Press the STEP switch</p> <p>26. Set the SPO switch off (last step in zeroing core)</p> <p>27. Reload operating TASK (table 7-1)</p> <p>28. Set data switch 2 on</p> <p>29. Set data switch 15 on</p> <p>30. Ensure that all other console switches are in the off position</p> <p>31. Press the console STEP switch</p>	<p>15</p>	<p>4.1</p>

Table 7-8. Building the Skeleton (Cont.)

Procedure Step	Page Reference	Paragraph
<p>32. Observe the following message: PLACE TASK DECK IN CARD HOPPER</p> <p>33. Place the on-line skeleton TASK object deck, without the TASK high-core loader, in the card reader hopper</p> <p>34. Place the Skeleton Builder deck behind the skeleton TASK object deck. Place a blank card behind the Skeleton Builder deck, that is, behind the *CCEND card</p> <p>35. Press the console STEP switch</p> <p style="text-align: center;">Note</p> <p>If a skeleton has previously been built or if the disk pack has been previously defined as a nonprocess monitor pack (XEQ cards allowed), the following message is printed:</p> <p style="text-align: center;">DATA SW 0 ON SAVE ICL TABLE</p> <p>In this case, set data switch 0 as desired and press the console STEP switch.</p> <p>If the TASK skeleton I/O has changed size, ICLT may not be saved. Use DUP *DICLE to restore the interrupt core load table.</p>	<p style="text-align: center;">49</p> <p style="text-align: center;">60</p>	<p style="text-align: center;">4.7</p> <p style="text-align: center;">5.10.12</p>

Table 7-8. Building the Skeleton (Cont.)

Procedure Step	Page Reference	Paragraph
<p>36. If error message 0026 occurs, return to step 4</p> <p>37. Observe the following message: PUT SKL BUILD PROG IN CARD HOPPER</p> <p>38. Press the console STEP switch</p> <p>39. Ensure that data switch 0 is off, press the console STEP switch, and wait for the following message: DATA SW 0 ON TO ABORT SKEL</p> <p style="text-align: center;">Note</p> <p style="text-align: center;">The abort option should not be selected.</p> <p>If any error messages occur before this message, correct the errors (see appendix A) and return to step 4</p>		
<p>40. Check the skeleton map to be sure that the skeleton has been correctly built. If the skeleton is incorrect, set data switch 0 on, press the console STEP switch, correct the errors, and return to step 4</p>	121	4.7
<p>41. If the skeleton is correct, press the console STEP switch. When the skeleton is transferred from core to disk, one of the following messages is printed:</p>		

Table 7-8. Building the Skeleton (Cont.)

Procedure Step	Page Reference	Paragraph
<p>SKB, SYDIR LD XQ (Skeleton is on disk and executable)</p> <p>SKB, SYDIR LD NX (Skeleton is on disk but contains level 1 errors)</p> <p>SKB, SYDIR NL NX (loading has been aborted)</p> <p>42. Press the console STEP switch</p> <p>43. Set data switch 0 on</p> <p>44. Set the RUN-IDLE switch to IDLE</p> <p>45. Ensure that all the switches in the bottom row on the control panel, beginning with the REGISTER SELECT switches, are in the upper position</p> <p>46. Ensure that all the data switches are off</p> <p>47. Press the ENTER switch</p> <p>48. Set the RUN-IDLE switch to RUN</p> <p>49. Press the STEP switch</p> <p style="text-align: center;">Note</p> <p>Steps 43 through 49 constitute a return to TASK.</p>		

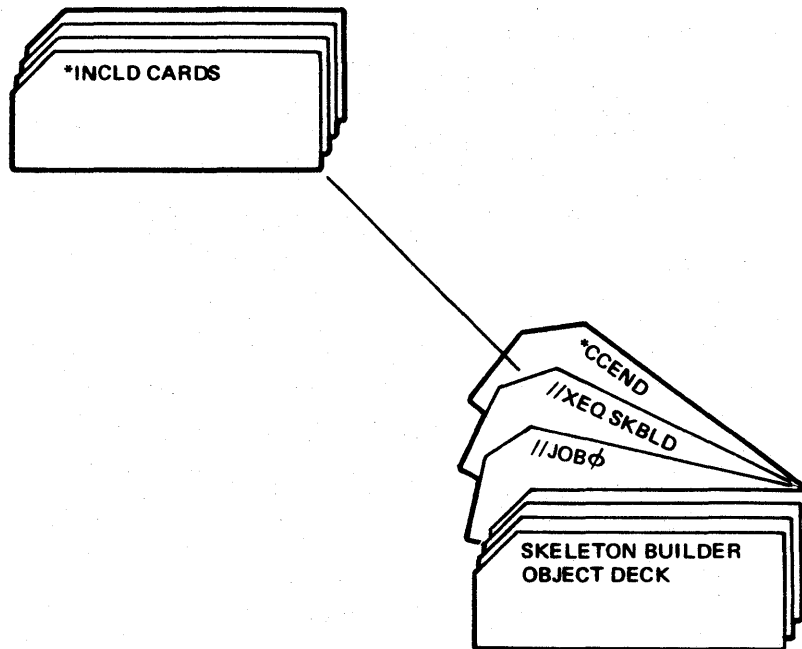


Figure 7-8. Skeleton Builder Object Deck and Control Cards

Table 7-9. Compiling Process Programs

Procedure Step	Page Reference	Paragraph
<ol style="list-style-type: none"> 1. Place the stacked input programs in the card reader hopper 2. Ready the reader 3. Set data switch 7 on 4. Press the CONSOLE INTERRUPT switch on the computer console 5. If errors occur, correct the errors (see appendix A) and stack the corrected jobs behind the last job in the card reader hopper. If the errors must be corrected before continuing with the remaining jobs, disable the reader by pressing STOP on the reader, set up the corrected jobs in the proper sequence in the reader, press the STEP switch on the computer console, and return to step 3 		

Table 7-10. Building Process Core Loads

Procedure Step	Page Reference	Paragraph
<p>1. Place the core load builder cards in the card reader hopper</p> <p>2. Ready the reader</p> <p>3. Set data switch 7 on</p> <p>4. Ensure that all other console switches are in the off position</p> <p>5. Press the CONSOLE INTERRUPT switch on the computer console</p> <p>6. Wait for the core load builder sign-on message as follows: CLB, BUILD NAME</p>	40	
<p>7. Check the core load map. If any errors have occurred that require reassembling and/or compiling, go to table 7-9</p>	119	5.7, 5.8
<p>8. If any errors have occurred because of improperly punched control cards, or if the core load does not meet requirements, correct the control cards and return to step 1</p>	50	4.7, 5.7
<p>9. Wait for the sign-off message for each core load, which is one of the following: CLB, LNAME LD XQ (core load is on disk and executable) CLB, LNAME LD NX (core load is on disk but normally not executable) CLB, LNAME NL NX (loading has been aborted)</p>		

Table 7-11. On-Line Cold Start

Procedure Step	Page Reference	Paragraph
<ol style="list-style-type: none"> 1. Set the RUN-IDLE switch to IDLE (first step in zeroing core) 2. Set the HALT switch on 3. Unlock the WSPS switch by turning it clockwise 4. Set the SPO switch on 5. Load the ZAP card, supplied with the system decks (see figure 7-3) 6. Ready the reader 7. Set all data switches off 8. Set REGISTER SELECT switches 4 and 8 on 9. Press RESET 10. Press ENTER 11. Set the HALT switch off 12. Press the IPL switch 13. Set the HALT switch on 14. Set REGISTER SELECT switches 4 and 8 off 15. Press RESET 16. Press ENTER 17. Set the RUN-IDLE switch to RUN 18. Press the STEP switch 		

Table 7-11. On-Line Cold Start (Cont.)

Procedure Step	Page Reference	Paragraph
19. Set the SPO switch off (last step in zeroing core)		
20. Place the system pack on any drive defined as part of the system		
21. Place the cold start loader cards (for the physical drive number corresponding to the physical drive on which the system pack is loaded) followed by a cold start name card and a blank card in the card reader hopper (figure 7-9). The cold start loader cards are shown in figure 7-10	42	4.9, 6.4
22. If the skeleton is to be storage protected, that is, column 14 in the *CLDST card is punched, turn on the console WSPS switch	42	6.4
23. Ready the reader		
24. Select the loading address in the data switches (first step in program load)		
25. Set the RUN-IDLE switch to IDLE		
26. Set the HALT switch on		
27. Set REGISTER SELECT switches 4 and 8 on		
28. Press RESET		
29. Press ENTER		
30. Set the HALT switch off		
31. Press the IPL switch		

Table 7-11. On-Line Cold Start (Cont.)

Procedure Step	Page Reference	Paragraph
<p>32. Set REGISTER SELECT switches 4 and 8 off</p> <p>33. Set the HALT switch on</p> <p>34. Press RESET</p> <p>35. Press ENTER</p> <p>36. Set the RUN-IDLE switch to RUN</p> <p>37. Press the STEP switch (last step in program load)</p> <p>38. If error messages occur, correct the errors (see appendix A)</p> <p>39. If the *CLDST card contains a punch in column 14, the system skeleton in core is storage protected and the following message is printed:</p> <p style="padding-left: 40px;">TURN OFF WRITE STORAGE PROTECT SWITCH</p> <p>40. If the following message is printed:</p> <p style="padding-left: 40px;">TURN ON WRITE STORAGE PROTECT SWITCH</p> <p>return to step 1</p> <p>41. Set the WSPS switch off</p> <p>42. Press the console STEP switch</p> <p>43. If the *CLDST card contains a 1 in column 16 (clock option), wait for the following message:</p> <p style="padding-left: 40px;">ENTER TIME THROUGH DATA SWITCHES</p>	<p>42</p> <p>42</p>	<p>6.4</p> <p>6.4</p>

Table 7-11. On-Line Cold Start (Cont.)

Procedure Step	Page Reference	Paragraph
<p>44. Enter the time in hexadecimal: the hours in switches 0 through 7 and the minutes in switches 8 through 15</p> <p>45. Press the console STEP switch. The time is read from the data switches, converted to hours and thousandths of an hour, and the following message is printed:</p> <p style="padding-left: 40px;">TIME ENTERED WAS XX.XXX HOURS</p> <p>The first process core load is called, and the TSS system runs under control of the System Director</p> <p>46. To initialize time sharing, turn on data switch 7 and press the CONSOLE INTERRUPT switch. The on-line system is now ready for operation</p>		

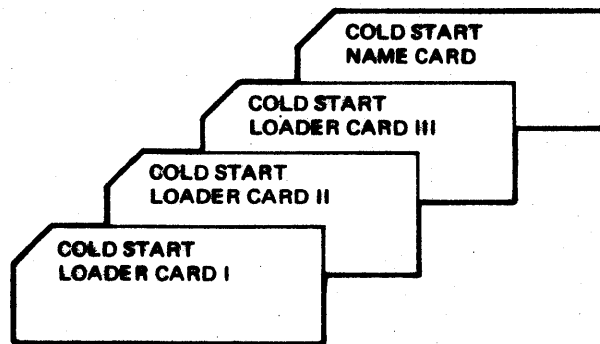


Figure 7-9. On-Line Cold Start

7.3.12 Building a Nonprocess Monitor Disk Pack

This procedure is presented with the assumption that the system decks have been loaded and the define configuration function has been executed, and that the TASK program has been loaded in core. If the system decks have not been loaded, see paragraph 7.3.2. If the configuration has not been defined, see paragraph 7.3.5.

To build a nonprocess monitor disk pack, perform the steps in table 7-13. This procedure is applicable for initial system generation, when TASK is already in core, and for building a nonprocess monitor pack when TASK is not in core.

7.3.13 Off-Line Cold Start

To perform an off-line cold start, perform the steps listed in table 7-14.

Table 7-13. Building a Nonprocess Monitor Disk Pack

Procedure Step	Page Reference	Paragraph
<p>1. If TASK is in core, proceed to step 2. If TASK is not in core, load TASK as directed in table 7-1</p> <p>2. Set the HALT switch on (first step in restarting TASK)</p> <p>3. Set the RUN-IDLE switch to IDLE</p> <p>4. Set the HALT switch off</p> <p>5. Ensure that the switches in the bottom row on the computer console, beginning with the REGISTER SELECT switches, are in the upper position</p> <p>6. Set all data switches off</p> <p>7. Press the ENTER switch</p> <p>8. Set the RUN-IDLE switch to RUN</p> <p>9. Press the STEP switch. Wait for the TASK messages to be printed</p> <p>10. Set data switches 0 and 15 on to select the absolute loader</p>	<p>15</p>	<p>4.1</p>
<p>11. Place the TASK disk loader (part of the utility package supplied with the system), followed by the operating TASK deck without the high-core loader, in the card reader hopper</p> <p>12. Ready the reader</p> <p>13. Press the STEP switch on the computer console</p> <p>14. If any errors occur, correct the errors as directed in appendix A</p>	<p>29</p>	<p>4.5.3</p>

Table 7-14. Off-Line Cold Start

Procedure Step	Page Reference	Paragraph
<ol style="list-style-type: none"> 1. Set the RUN-IDLE switch to IDLE (first step in zeroing core) 2. Set the HALT switch on 3. Unlock the WSPS switch by turning it clockwise 4. Set the SPO switch on 5. Load the ZAP card, supplied with the system decks (see figure 7-3) 6. Ready the reader 7. Set all data switches off 8. Set REGISTER SELECT switches 4 and 8 on 9. Press RESET 10. Press ENTER 11. Set the HALT switch off 12. Press the IPL switch 13. Set the HALT switch on 14. Set REGISTER SELECT switches 4 and 8 off 15. Press RESET 16. Press ENTER 17. Set the RUN-IDLE switch to RUN 18. Press the STEP switch 		

Table 7-14. Off-Line Cold Start (Cont.)

Procedure Step	Page Reference	Paragraph
<p>19. Set the SPO switch off (last step in zeroing core)</p> <p>20. Place the proper cold start loader cards (figure 7-10) followed by a cold start TASK name card and the stacked nonprocess jobs in the card reader hopper (see figure 7-11)</p> <p>21. Ready the reader</p> <p>22. Select the loading address in the data switches (first step in program load)</p> <p>23. Set the RUN-IDLE switch to IDLE</p> <p>24. Set the HALT switch on</p> <p>25. Set REGISTER SELECT switches 4 and 8 on</p> <p>26. Press RESET</p> <p>27. Press ENTER</p> <p>28. Set the HALT switch off</p> <p>29. Press the IPL switch</p> <p>30. Set REGISTER SELECT switches 4 and 8 off</p> <p>31. Set the HALT switch on</p> <p>32. Press RESET</p> <p>33. Press ENTER</p> <p>34. Set the RUN-IDLE switch to RUN</p> <p>35. Press the STEP switch (last step in program load). The system is now ready for off-line operation</p>	<p>42</p>	<p>4.9, 6.4</p>

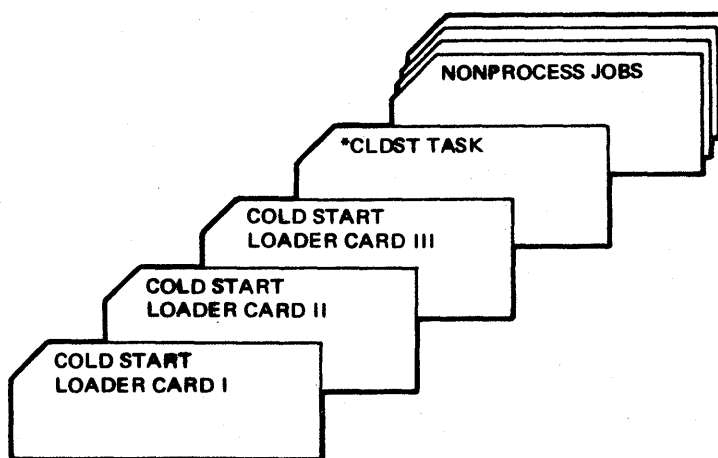


Figure 7-11. Off-Line Cold Start

APPENDIX A – SYSTEM ERROR MESSAGES

The system error messages and recovery procedures--where applicable--are presented in the following tables.

A.1 TASK ERROR ALERT CONTROL PROGRAM ERRORS

The EAC error messages are divided into four groups:

I/O errors

FORTTRAN execution errors

Internal errors

TASK errors

All EAC error messages are listed in table A-1.

A.1.1 I/O Errors

The format for I/O error messages produced by TASK is

ERROR code ep₁ ep₂ ep₃ ep₄

where

code - identifies the specific error; $0000 \leq \text{code} \leq 0063$.

ep₁ - if the error is a CALL error, ep₁ is the address of the invalid call; otherwise, ep₁ is the address of the device table for the hardware device that caused the difficulty.

ep₂ - beginning address of the interrupt level work area for the level of the call. (ep₂ is not applicable to TYPEN.)

ep₃ - address of an invalid call on any but a CALL error. (ep₃ is not applicable to TYPEN.)

ep₄ - applicable only when device is a magnetic tape unit, in which case it specifies the device status word (DSW).

A.1.2 FORTRAN Execution Errors

The format of error messages for errors detected during execution of a FORTRAN program is

ERROR code

where code identifies the specific error; $0064 \leq \text{code} \leq 006D$.

Table A-1. TASK Error Alert Control Errors

Error Code	Type of Error	Meaning	Recovery
0000	I/O	Illegal call to/from teletypewriter	Restart TASK.
0001	I/O	Teletypewriter not ready	Correct and continue.
0003	I/O	Teletype keyboard not ready	Correct and continue.
0004	I/O	Storage protection violation	Reload TASK and restart.
0005	I/O	Teletype keyboard parity error	Restart TASK.
0006	I/O	Teletypewriter parity error	Restart TASK.
0008	I/O	Invalid message on disk	Restart TASK; correct and continue.
000A	I/O	Card reader or card punch invalid call	Restart TASK.
000B	I/O	Card reader last card indicator	Remove cards left in reader. Press console STEP. Place remainder of cards to be read in reader hopper, nonprocessed cards first, and press reader START.
000C	I/O	Card reader or card punch parity error	Correct and continue; restart TASK.
000D	I/O	Card reader storage protection violation	Restart TASK; reload TASK.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
000E	I/O	Card reader or card punch feed check	Correct and continue; restart TASK.
000F	I/O	Card reader or card punch data overrun	Correct and continue; restart TASK.
0010	I/O	Card reader or card punch check	Correct and continue.
0011	I/O	// Ø card illegally read	Correct and continue; restart TASK.
0013	I/O	Card reader or card punch not ready	Correct and continue.
0014	I/O	Paper tape reader or punch invalid call	Restart TASK.
0015	I/O	Paper tape punch parity error	Restart TASK.
0016	I/O	Paper tape reader not ready	Correct and continue.
0017	I/O	Paper tape punch not ready	Correct and continue.
0018	I/O	Paper tape reader parity error	Restart TASK; correct and continue.
0019	I/O	Paper tape reader or punch storage protect violation	Restart TASK; reload TASK.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
001E	I/O	Disk invalid call	Restart TASK.
001F	I/O	Disk not ready	Correct and continue.
0020	I/O	Disk data overrun	Restart TASK.
0021	I/O	Disk write select	Reload TASK.
0022	I/O	Disk data error	Restart TASK.
0023	I/O	Disk storage protect error	Restart TASK; reload TASK.
0024	I/O	Disk parity error	Restart TASK.
0025	I/O	Disk invalid address	Restart TASK; reload TASK.
0026	I/O	Disk file protect error	Restart TASK.
0027	I/O	Disk hardware/program malfunction (internal or lost interrupt)	Restart TASK.
0028	I/O	Plotter invalid call	Restart TASK.
0029	I/O	Plotter parity error	Restart TASK.
002A	I/O	Plotter not ready	Correct and continue.
0032	I/O	Line printer invalid call	Restart TASK.
0036	I/O	Line printer parity error (i. e. , print operation requested, but data not in printer code)	Restart TASK; correct and continue.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
0037	I/O	Line printer not ready	Correct and continue.
003C	I/O	Invalid call	Restart TASK.
003D	I/O	Storage protect violation	Restart TASK.
003E	I/O	Parity control error	Restart TASK.
003F	I/O	Parity data error	Restart TASK.
0040	I/O	Overlap conflict	Correct and continue; restart TASK.
0041	I/O	Intermediate table interrupt (error code passed to user's special condition routine)	Correct and continue.
0042	I/O	Any error (error code passed to user's special condition routine)	Correct and continue.
0045	I/O	Comparator violation (error code passed to user's special condition routine)	Correct and continue.
0046	I/O	Invalid call	Restart TASK.
0047	I/O	Parity error	Restart TASK.
0048	I/O	Storage protect violation	Restart TASK; reload TASK.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
0049	I/O	Intermediate table interrupt (error code passed to user's special condition routine)	Correct and continue.
0050	I/O	Invalid call	Restart TASK.
0051	I/O	Parity error	Restart TASK.
0052	I/O	Intermediate table interrupt (error code passed to user's special condition routine)	Correct and continue.
005A	I/O	Magnetic tape unit invalid call	Restart TASK.
005C	I/O	Magnetic tape unit storage protect violation	Restart TASK.
005D	I/O	Magnetic tape unit command reject	Restart TASK; correct and continue.
005E	I/O	Magnetic tape unit excessive tape errors	Restart TASK; correct and continue.
005F	I/O	Magnetic tape unit tape error	Restart TASK; correct and continue.
0063	I/O	Magnetic tape unit end of tape	Restart TASK; correct and continue.
0064	FORTRAN	Illegal address computed in an indexed store	Restart TASK.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
0065	FORTRAN	Illegal integer used in a Computed GO TO statement	Restart TASK.
0066	FORTRAN	File not defined	Restart TASK.
0067	FORTRAN	Requested number of records too large, zero, or negative	Restart TASK.
0068	FORTRAN	Input record in error or illegal conversion	Restart TASK.
0069	FORTRAN	Range of numerical values in error	Restart TASK.
006A	FORTRAN	Output field too small to contain number	Correct and continue.
006B	FORTRAN	Illegal unit reference	Restart TASK.
006C	FORTRAN	Requested record length exceeds buffer capacity	Restart TASK.
006D	FORTRAN	Working storage area insufficient for define files	Restart TASK.
0096	FORTRAN	Illegal unit reference. Unit not defined in I/O unit table, on IOCS card, or for unedited I/O	Restart TASK.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
0097	FORTRAN	Read list exceeds length of write list. List in READ statement is longer than list in corresponding WRITE statement	Restart TASK.
0098	FORTRAN	Record does not exist for read list element. Last physical record of logical record has been exhausted	Restart TASK.
1000	Internal	Channel Address Register check	Restart TASK; reload TASK.
2000	Internal	Storage protect violation	Restart TASK; reload TASK.
4000	Internal	Parity error	Reload TASK.
8000	Internal	Operation code violation	Restart TASK.
F001	TASK	Monitor XEQ tried when not allowed (i. e., disk pack not defined as an off-line system, error code of F006 was previously given, or program attempting execution is not in LET/FLET	Restart TASK.

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
F002	TASK	No sectors on disk for buffering of Teletype messages when TASK requires it	Reload TASK.
F003	TASK	Invalid word count or sector address for program to be loaded to disk	Restart TASK.
F004	TASK	TASK conversion routines called with a negative or zero word count	Restart TASK.
F005	TASK	Mode switch on TRACE when the trace program is not in TASK	Correct and continue.
F006	TASK	TASK in core is not the same one defined on disk for off-line systems. No XEQ will be allowed.	Reload TASK; correct and continue.
F007	TASK	Checksum or sequence error in absolute deck being loaded by TASK. ep_1 is the sequence number of the card in error (hexadecimal).	Correct sequence or checksum error and reload deck starting with card sequence number printed in ep_2 .

Table A-1. TASK Error Alert Control Errors (Cont.)

Error Code	Type of Error	Meaning	Recovery
F008	TASK	Too many defective cylinders on disk to allow skeleton build	Reload TASK.
F009	TASK	Logical disk drive named in ep ₁ is not ready. If console STEP is pressed before the drive becomes ready, the drive will be taken off-line.	Correct and continue.
F00A	TASK	Logical disk drive named in ep ₁ has been taken off-line because it is not ready (see error message F009).	Correct and continue.
FFFF	TASK	An error recovery procedure of "correct and continue" was attempted when not allowed.	Restart TASK; reload TASK.

A.1.3 Internal Errors

The format of this group of error messages is

ERROR code ep₁ ep₂ ep₃ ep₄

where

code - identifies the specific error; $1000 \leq \text{code} \leq \text{F000}$.

ep₁ - contents of the I-register at the time the error occurred. (Usually the instruction that caused the error will be ep₁ - 1 or ep₁ - 2.)

ep₂ - contents of index register 1 at the time of the error.

ep₃ - contents of index register 2 at the time of the error.

ep₄ - contents of index register 3 at the time of the error.

A.1.4 TASK Errors

TASK errors usually denote incorrect system operating procedures. The format for this group of messages is

ERROR code e₁

where

code - identifies the specific error; $\text{F001} \leq \text{code} \leq \text{F006}$.

e₁ - defined in table A-1 with applicable message.

A.2 ASSEMBLER ERROR MESSAGES

System error messages produced during the assembly of a source program have the format:

Ann message

where

Ann - error code; the letter A denotes an Assembler message, and nn is merely a 2-digit sequence number.

message - character string that describes the situation.

The Assembler error messages are listed in table A-2.

Table A-2. Assembler Error Messages

Error Message	Meaning	Recovery
<p>A01 MINIMUM W.S. NOT AVAILABLE-- ASSEMBLY TERMINATED</p>	<p>Available nonprocess working storage is less than the number of overflow sectors specified plus 1.</p>	<p>1. Reduce the number of overflow sectors specified (number specified = 0 if *OVERFLOW SECTORS control card is not used).</p> <p>2. If more than one drive is available on the system, specify drive on JOB card with most nonprocess working storage available.</p>
<p>A02 SYMBOL TABLE EXCEEDS SPECIFIED OVERFLOW</p>	<p>Actual number of sectors of symbol table overflow is greater than the number of overflow sectors allocated.</p>	<p>Use the *OVERFLOW SECTORS control card to increase the number of sectors specified (maximum 32).</p>
<p>A03 DISK OUTPUT EXCEEDS W.S.</p>	<p>Intermediate output in pass 1 or final DSF output in pass 2 is greater than nonprocess working storage less the number of overflow sectors specified.</p>	<p>1. If error occurs in pass 1, the assembler will wait at: 8K = 1B5E 16K = 3B5E 32K = 7B5E When console START is pressed, the assembly will automatically be continued in the *TWO PASS MODE.</p> <p>2. If error occurs in pass 2, reduce the number of overflow sectors specified or specify drive on JOB card with the most nonprocess working storage available if more than one drive is available.</p>

Table A-2. Assembler Error Messages (Cont.)

Error Message	Meaning	Recovery
<p>A04 SAVE SYMBOL TABLE INHIBITED</p>	<p>With *SAVE SYMBOL TABLE option specified:</p> <ol style="list-style-type: none"> 1. Program is a relocatable assembly. 2. Program contains assembly errors.† 3. Source program causes more than 100 symbols to be present in the system symbol table. 	<ol style="list-style-type: none"> 1. Use ABS card and reassemble. 2. Correct source program errors and reassemble. 3. Reduce number of symbols in program and reassemble.
<p>A05 MAINLINE PROGRAM WITH NO NAME</p>	<p>Mainline program just assembled had no name specified on // ASM card.</p>	<p>Punch program name into name field of // ASM card and reassemble.</p>
<p>A06 MORE THAN 25 ERRORS IN ORG, BSS, OR EQU STATEMENTS-- ASSEMBLY TERMINATED</p>	<p>Of the specified statements (including BES) 25 operands were undefined in pass 1 and defined in pass 2. An attempt has been made, for example, to ORG ahead or to equate a symbol with a forward reference.</p>	<p>If LIST option is used or if a partial list deck has been punched, look for the error flag U* in print position or column 18 in statements of the specified type. If forward references have been attempted, these must be corrected before the program is reassembled.</p>
<p>†Specific assembly-time errors are presented in table A-3.</p>		

Table A-2. Assembler Error Messages (Cont.)

Error Message	Meaning	Recovery
A07 LOAD BLANK CARDS	A card containing a nonblank column in columns 1 through 71 has been read while punching the symbol table (as a result of a *PUNCH SYMBOL TABLE control card).	Place blank cards in the hopper of the card punch. Press punch START and console STEP.
A08 CONTROL RECORD READ--ASSEMBLY TERMINATED	A //control record has been read by the assembler. The assembler passes this card along to the Supervisor before terminating the assembly. Loading and DUP operations are inhibited.	Ensure that all cards including the END card are in the source deck. Reassemble.

Table A-3. Assembler Error Detection Codes

Code	Cause	Assembler Action
A	<u>Address Error.</u> Attempt made to specify displacement field, directly or indirectly, outside range of -128 to +127.	Displacement set to zero.
C	<u>Condition Code Error.</u> Character other than +, -, Z, E, C, or O detected in first operand of short branch or second operand of long BSC, BOSC, or BSI statement.	Displacement set to zero.
F	<u>Format Code Error.</u> Character other than L, I, X, or blank detected in column 32; or L or I format specified for instruction not valid in that form.	Instruction processed as if L format were specified, unless that instruction is valid only in short form, in which case it is processed as if the X format were specified.
L	<u>Label Error.</u> Invalid symbol detected in label field.	Label ignored.
M	<u>Multiply Defined Label.</u> Duplicate symbol encountered in label field or in operand.	First occurrence of symbol in label field defines its value; subsequent occurrences of the symbol in label field cause a multiply defined indicator to be inserted in Symbol Table entry (bit 0 of first word).

Table A-3. Assembler Error Detection Codes (Cont.)

Code	Cause	Assembler Action
R	<p><u>Relocation Error.</u> (1) Expression does not have valid relocation.</p> <p>(2) Nonabsolute displacement specified.</p> <p>(3) Absolute origin specified in relocatable program.</p> <p>(4) Nonabsolute operand specified in BSS or BES.</p> <p>(5) Nonrelocatable operand in END statement of relocatable main program.</p> <p>(6) ENT operand nonrelocatable</p>	<p>Expression set to zero.</p> <p>Displacement set to zero.</p> <p>Origin ignored.</p> <p>Operand assumed to be zero.</p> <p>Entry assumed to be relative zero.</p> <p>Statement ignored.</p>
S	<p><u>Syntax Error.</u> (1) Invalid expression (e. g. , invalid symbol, adjacent operators, illegal constant).</p> <p>(2) Main program entry point not specified in END operand.</p> <p>(3) Incorrect syntax in EBC statement (e. g. , no delimiter in card column 35 or zero character count).</p> <p>(4) Invalid label in ENT or ISS operand.</p>	<p>Expression set to zero.</p> <p>Entry assumed to be relative zero.</p> <p>Location counter incremented by 17.</p> <p>Statement ignored</p>
T	<p><u>Tag Error.</u> Card column 33 contains character other than blank, 0, 1, 2, or 3 in instruction statement.</p>	<p>Tag of zero assumed.</p>

Table A-3. Assembler Error Detection Codes (Cont.)

Code	Cause	Assembler Action
U	<u>Undefined Symbol.</u> Undefined symbol encountered in expression.	Expression set to absolute zero.
O	<u>Operation Code Error.</u> (1) Operation code not recognized. (2) ISS, ILS, ENT, LIBR, SPR, EPR, or ABS incorrectly placed.	Statement ignored and Location Counter incremented by 2. Statement ignored.

A.3 FORTRAN ERROR MESSAGES

Error messages produced by FORTRAN during a compilation are listed in table A-4. If both EAC and FORTRAN messages are output to the line printer, it is possible that an EAC message may overprint the previous FORTRAN message. The user can prevent this overprinting by employing either of two methods:

1. Assign the teletypewriter as the EAC output device.
2. Provide a line space after each FORTRAN printout (e.g., a slash at the end of each FORMAT statement or a 1H+ at the beginning of each FORMAT statement).

Table A-4. FORTRAN Error Codes

Error Number	Cause of Error
C1	Statement number contains a nonnumeric character.
C2	Maximum number (five) of continuation cards exceeded, or continuation card out of sequence.
C3	Syntax error in CALL LINK or CALL EXIT statement, or CALL LINK or CALL EXIT statement in process program.
C4	Undeterminable, misspelled, or incorrectly formed statement.
C5	Statement out of proper sequence.
C6	Statement following STOP, RETURN, CALL LINK, CALL EXIT, GO TO, IF, or TSS CALL statement should but does not have a statement number.
C7	Name consists of more than five characters, or name starts with a nonalphabetic character.
C8	Incorrect or missing subscript within dimension information (DIMENSION, COMMON, REAL, or INTEGER).
C9	Duplicate statement number.
C10	Syntax error in COMMON statement.
C11	Duplicate name in COMMON statement.
C12	Syntax error in FUNCTION or SUBROUTINE statement.
C13	COMMON statement contains parameter (dummy argument).
C14	SUBROUTINE or FUNCTION statement contains a name twice as a parameter.
C15	*IOCS control record in a subprogram.
C16	DIMENSION statement contains a syntax error.
C17	DIMENSION statement contains a subprogram name.
C18	Name dimensioned more than once, or not dimensioned on first appearance of name.

Table A-4. FORTRAN Error Codes (Cont.)

Error Number	Cause of Error
C19	REAL, INTEGER, or EXTERNAL statement contains a syntax error.
C20	REAL or INTEGER statement contains a subprogram name.
C21	Name in EXTERNAL statement is also in a COMMON or DIMENSION statement.
C22	Reference to IFIX or FLOAT function in EXTERNAL statement.
C23	Invalid real constant.
C24	Invalid integer constant.
C25	More than 15 dummy arguments, or statement function argument list contains duplicate dummy arguments.
C26	A subscript expression is missing a right parenthesis.
C27	FORMAT statement contains a syntax error.
C28	Statement number missing from FORMAT statement.
C29	Field width specification greater than 145.
C30	In a FORMAT statement the E or F conversion specifies w greater than 127, d greater than 31, or d greater than w (where w is an unsigned integer constant specifying the total field length of the data, and d is an unsigned integer constant specifying the number of decimal places to the right of the decimal point).
C31	EQUIVALENCE statement contains a subscript error.
C32	A statement function contains a subscripted variable.
C33	Subscript expression incorrectly formed.
C34	Subscript expression contains undefined variable.
C35	A subscript expression contains some number of subscripts that does not agree with the dimension information.

Table A-4. FORTRAN Error Codes (Cont.)

Error Number	Cause of Error
C36	Invalid arithmetic statement or variable; or, in a FUNCTION subprogram, the left side of an arithmetic statement is a dummy argument (or in COMMON).
C37	IF statement contains a syntax error.
C38	IF statement contains an invalid expression.
C39	CALL statement contains a syntax error or invalid simple argument.
C40	CALL statement contains an invalid expression.
C41	A statement function contains an invalid expression to the left of an equals sign.
C42	A statement function contains an invalid expression to the right of an equals sign.
C43	An IF, GO TO, or DO statement number is missing, invalid, incorrectly placed, or else it is the same as a FORMAT statement number.
C44	READ or WRITE statement contains a syntax error.
C45	A mainline program contains a READ or WRITE statement but *IOCS record is missing.
C46	A READ or WRITE statement does not contain a FORMAT statement number or the FORMAT statement number is incorrect.
C47	Syntax error in input/output list; or an invalid list element; or, in a FUNCTION subprogram, the input list element is a dummy argument or is in COMMON.
C48	GO TO statement contains a syntax error.
C49	Index of Computed GO TO statement is missing, invalid, or not preceded by a comma.

Table A-4. FORTRAN Error Codes (Cont.)

Error Number	Cause of Error
C50	A mainline program contains a *TRANSFER TRACE or *ARITHMETIC TRACE control record present with no *IOCS control record.
C51	DO statements are incorrectly nested; or the terminal statement of the associated DO statement is a GO TO, IF, RETURN, FORMAT, STOP, PAUSE, DO, or TSS CALL statement.
C52	Number of nested DO statements exceeds maximum allowed (25).
C53	DO statement contains a syntax error.
C54	DO statement has zero as initial value.
C55	In a FUNCTION subprogram the index variable of DO statement is a dummy argument or is in COMMON.
C56	BACKSPACE statement contains a syntax error.
C57	REWIND statement contains a syntax error.
C58	END FILE statement contains a syntax error.
C59	A STOP statement occurs in a process program, or a STOP statement contains a syntax error.
C60	PAUSE statement contains a syntax error.
C61	STOP or PAUSE statement contains an integer constant greater than 9999.
C62	Last executable statement before END statement is not a STOP, GO TO, IF, CALL LINK, CALL EXIT, RETURN, or TSS CALL statement.
C63	Statement contains more than 15 different subscript expressions.
C64	Because of compiler expansion of subscript expressions or compiler addition of generated temporary storage locations, statement has become too long to be scanned.

Table A-4. FORTRAN Error Codes (Cont.)

Error Number	Cause of Error
C65†	All variables in an EQUIVALENCE list† are undefined.
C66†	Variable made equivalent to an element of an array in such a manner as to cause the array to extend beyond the origin of the COMMON area.†
C67†	Two variables or array elements in COMMON are equated, or the relative locations of two variables or array elements are assigned more than once (directly or indirectly).†
C68	EQUIVALENCE statement contains a syntax error; or an EQUIVALENCE list contains an illegal variable name.
C69	Subprogram does not contain a RETURN or TSS CALL statement; or a mainline program contains a RETURN statement.
C70	A mainline program that contains disk READ, WRITE, or FIND statements has no DEFINE FILE.
C71	DEFINE FILE contains a syntax error.
C72	Duplicate DEFINE FILE statements, maximum allowed number (75) of DEFINE FILE statements exceeded, or DEFINE FILE statement occurs in subprogram.
C73	Record number of READ, WRITE, or FIND statement contains a syntax error,
C74	INSKEL COMMON referenced with two-word integers.
C75	DATA statement contains a syntax error.
C76	In a DATA statement the names and constants are not one to one.
C77	DATA statement contains mixed mode values.
C78	DATA statement contains an invalid Hollerith constant.
C79	DATA statement contains an invalid hexadecimal specification.
† The detection of an error identified by code 65, 66, or 67 prevents any subsequent detection of any of these three errors.	

Table A-4. FORTRAN Error Codes (Cont.)

Error Number	Cause of Error
C80	DATA statement contains a variable that is not used elsewhere in the program.
C81	COMMON variable loaded with a DATA specification.
C82	DATA statement too long.
C83	TSS CALL statement appears illegally (CALL INTEX, CALL BACK, CALL DPART, CALL CHAIN, or CALL VIAQ in a nonprocess program; CALL VIAQ, CALL CHAIN, or CALL BACK in a process subroutine).

A.4 DISK UTILITY PROGRAM ERROR MESSAGES

Error Messages produced by DUP are listed in table A-5.

Table A-5. DUP Error Messages

Error Code	Message	Meaning/Action	Recovery Code [†]
D01	INVALID CNTRL CD	DUP or Monitor control card invalid.	U
D02	PGM ALRDY IN TMP	The program named already has entries in temporary LET. Execute indicator is not set.	U
D03	DR USED THIS JOB	Control card specified a disk drive that is already in use with this job; therefore, only the label was changed.	R
D04	DR 0 NOT MON. PK	Disk drive zero was specified for use, but *DLABL control card did not identify it as a monitor pack; therefore, the new label is placed on drive zero.	R
D05	ABORT SIGN OFF	When an unrecoverable error is encountered, this message is produced following the appropriate error message.	U
D06	DRIVE NOT IN USE	1. Disk drive specified for this job is not in use. 2. A dump of LET/FLET area was requested from a drive not in use.	U

[†]Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code†
D07	DR GT MAX FOR SYSTEM	The source drive specified is greater than the maximum allowed for the system.	U
D09	NO CALL IN xxxxx	No call appears in core load xxxxx for the core load being changed.	R
D10	CD CHKSUM ERROR	Checksum error occurred while binary program cards were being read.	U
D11	NEED BLANK CARDS	Insufficient supply of blank cards for dumping to cards. Place new supply of blank cards in punch hopper, press punch START switch, and punching will continue.	R
D12	MON CRD WAS READ	A monitor control card was read when a DUP card was expected. The specified monitor function will be performed, and the previous DUP store function will be terminated.	U

†Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code†
D20	INVALID CHARACTER	The buffer holding the printer output contains invalid EBCDIC characters. The printer will leave one blank space for each invalid character. After the message has been printed, the function will be terminated.	U
D21	INVALID PGM TYPE	<ol style="list-style-type: none"> 1. Following an assembly or compilation, an invalid type code was found in program header. 2. An *STORECI control card contained a program type code in column 11 that was not an M, C, I, or blank. No execute indicator (INOEX) is set, and the *STORECI function is aborted. 3. Replacement core load was not the correct type. 	U
D22	IL PGM HDR LNTH	A dump of a relocatable program was requested, but an illegal program header length was detected.	U

†Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code [†]
D23	DUMP LENGTH = ZERO	The size of the program or data area to be dumped was calculated to be zero. The error exists in the LET/FLET entry or, for a relocatable program, in the program header.	U
D24	CTRL CD NAME BAD	An invalid or blank name was found on a control card. 1. If found by DUMP1 or SEQCH program, function is aborted. 2. If found by SCONT or DLETE program, function may be allowed to continue.	U/R
D25	NAME NOT IN L/F	1. Routine to be removed was not on drive zero. 2. A search of LET/FLET for a particular program name was unsuccessful.	U
D28	NO SRCE DR SPCFD	Control card failed to specify a source drive. Source was indicated as nonprocess work storage.	U

[†]Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code†
D29	CNT FLD IN ERROR	Erroneous numeric field in control card. The count field either contains invalid characters or is located in wrong columns.	U
D31	INVALID INP CARD	This error occurs when any card other than a DUP control card is input to one of the store functions and is in error. Processing is terminated, and control is returned to the control card analyzing routine. The no execute indicator (INOEX) is set, and the program is not stored.	U
D32	NAME NOT PRIME	Control card contained a name that did not compare with prime entry point in binary deck. If no other entries with the same name exist in the table, the store function will continue to store program under prime entry point name	U
D34	NAME NOT IN TEMP	The temporary LET entries were searched for the name of the program to be stored, but it was not found.	U

†Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code [†]
D35	L/F FIND ILLEGAL	The name of the program to be stored already exists in LET/FLET for a particular drive. No execute indicator (INOEX) is set.	U
D38	NO DISK ROOM	<ol style="list-style-type: none"> 1. Sector address is out of range. The areas defined on CONFIG card exceed the space available. Requested size must be reduced. 2. Search of user area for permanent store, fixed area for FLET store, working storage for data store, or temporary storage area revealed insufficient room to store program. 3. Insufficient nonprocess work storage to allow dump operation to be completed. 4. This message is also produced when the specified drive is not on the system. 	U
D41	PNT EXCESS	Program name table exceeds allocated buffer space.	U

[†]Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code†
D43	TEMP ABORT	This message indicates that the disk pack is reaching the maximum number of programs the system can store. Suggested recourse: execute *DFINE PAKDK for that drive.	R
D44	L/F TABLES FULL	LET or FLET area is full. Increase size of LET/FLET or remove some entries.	U
D45	CORE LOAD NAMES NOT FOUND	This message precedes a list (10 per line) of core load names that are not found on disk.	R
D46	INOLD ON	This message notifies the user that the no-load indicator in the nonprocess communication area was set when control returned to DUP from the Core Load Builder, FORTRAN, or Assembler.	U
D47	NO PNT CMP	During the updating of core load program name table, the first entry in the table did not compare with the name placed in the nonprocess communication area from the *STORECI control card.	U
† Recovery codes are defined as follows: R = recoverable error allowing DUP to continue function. U = unrecoverable error causing DUP to abort the specific function.			

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code†
D50	NO SKEL ON DSK	No skeleton exists on disk.	U
D52	LEVEL/POS = L1/B1	Illegal level/bit position (i. e. , L1/B1) requested. This message is preceded by a D79 message for an illegal interrupt to show the interrupt level still calling the program. If this message does follow D79, the error is recoverable. Perform a DICLE function to enter the correct level and bit in the ICL table.	U/R
D75	PGM NAMES EQUAL	Names on the control card are the same.	U
D78	CL TYPE IN ERROR	The control card-specified type of core load (M, I, C, D, or blank) does not agree with the type indicated in FLET entry.	U
D79	PGM STILL CALLING	1. Other core loads have calls to a program that is to be deleted. The names of the calling programs are printed preceding this error message. 2. See also D52 error message.	U

†Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code†
D90	SKEL TOO LARGE	Size of skeleton is greater than object core. Change skeleton size or redefine object core to larger size.	U
D91	MESS NOT DEFINED	An M appeared on a CONFIG card, but message area not defined on any system disk. If system has no message buffer, remove the M entry from the card. Ascertain that the JOB card indicates all drives that are to be available to the system.	U
D92	INVALID FIELD(S)	The control card has some kind of error in format or content. See discussion of DUP control cards in paragraph 3.2.2.	U
D93	NO 'I' ON CONFIG CD	Control card contains "S" or "SX" but no "I". Add drive number and interrupt save length to the card.	U
D94	NO CLST OR ERPG	/CLST or .ERPG not found on any system drive. Add disk pack with these entries or ensure that all drives are indicated on JOB card.	U

†Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

Table A-5. DUP Error Messages (Cont.)

Error Code	Message	Meaning/Action	Recovery Code [†]
D95	PSBL CI PGM LOSS	<p>This *DFINE CONFG operation may cause some core image programs to be lost. Correct this condition by either of two methods:</p> <ul style="list-style-type: none"> ● Precede the *DFINE card with another *DFINE card to enlarge the core image area. ● Remove core image programs. 	U
D96	INSV GR THAN COR	<p>Defined interrupt save length exceeds variable core size. Check for:</p> <ul style="list-style-type: none"> ● Object core defined large enough. ● Skeleton defined too large. ● Interrupt save length defined too large. 	U
D99	SYST/HARDW ERROR	General error alarm; system or hardware error.	U

[†]Recovery codes are defined as follows:

R = recoverable error allowing DUP to continue function.

U = unrecoverable error causing DUP to abort the specific function.

A.5 SKELETON BUILDER ERROR MESSAGES

Error messages produced by the Skeleton Builder have the following format:

Kxx lprog rout reinc LEV.y

where

K identifies Skeleton Builder messages.

xx sequence number; $00_{16} \leq xx \leq FF_{16}$.

lprog name of the program being relocated.

rout routine specified within lprog.

reinc relocation increment of rout from lprog.

LEV.y denotes severity level of the error:

y = 0 - for information only; the Skeleton Builder is operating properly.

y = 1 - minor error; an error has occurred, but loading is continued.

y = 2 - major error; no loading possible; pass 2 will not be attempted.

y = 3 - severe error; abort immediately.

(Parameters lprog, rout, and reinc appear in the output only where required by the particular message.)

After the skeleton has been built, it can be moved to the skeleton area only when y is less than 2 (i. e., has the value 0 or 1).

The Skeleton Builder error messages are listed in table A-6.

A.6 SYSTEM LOADER ERROR MESSAGES

System Loader error messages consist of the identifying letter L and a two-digit sequence number. These messages are listed in table A-7.

Table A-6. Skeleton Builder Error Messages

Error Message	Meaning	Recovery
K01 LEV. 3	Load table overflow. This condition causes immediate termination of Skeleton Builder operation.	<ol style="list-style-type: none"> 1. Decrease number of subroutines. 2. Decrease number of in-core interrupt routines (see *INCLD in Non-process Monitor, paragraph 3.2.2).
K02 lprog LEV. 2	A disk format error has been found in program lprog.	Correct program on disk.
K03 rout LEV. 2	The relocatable program rout cannot be found in LET.	<ol style="list-style-type: none"> 1. Delete all references to rout from the system skeleton. 2. Store rout on disk and enter in LET.
K05 lprog rout reinc LEV.0,1,2	Within the skeleton the call to TSS subroutine rout in program lprog is invalid. The CALL statement is at relative location reinc of lprog.	Correct the CALL statement.

Table A-6. Skeleton Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
K06 lprog rout reinc LEV. 2	The required core load name in the list of the TSS program sequence change CALL to rout at relative location reinc does not occur in the EXTERNAL statement in lprog.	Correct lprog.
K07 lprog llbb LEV. 2	The in-core interrupt subroutine lprog has been designated to service llbb, which has been previously assigned to another in-core interrupt subroutine.	Correct the assignment of in-core interrupts on the *INCLD control card (paragraph 3.2.2).
K08 LEV. 2	The disk drive that was designated for temporary system usage does not contain enough nonprocess working storage space to build the skeleton.	<ol style="list-style-type: none"> 1. Change the temporary drive to another drive. 2. Delete programs from the user's area to provide sufficient nonprocess working space.
K09 LEV. 2	The maximum length of 255 words (85 entries) for the transfer vector (ETV) has been exceeded.	Decrease the number of LIBF subroutines included within the skeleton.
K0A lprog LEV. 3	Program lprog has a back-origin that would overlay part of the Skeleton I/O area.	Correct the lprog back-origin.

Table A-6. Skeleton Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
K0B LEV.2	The sum of the lengths of the skeleton and components exceeds the defined length.	Redefine the length of the skeleton or eliminate or shorten the skeleton components. This action will reduce the size of the skeleton.
K0C lprog iaccd LEV.1	The interrupt service subroutine lprog contains an interrupt service entry point, defined for IAC code (iaccd), which is not in the system master branch table. The IAC number (iaccd) is hexadecimal.	<ol style="list-style-type: none"> 1. Correct the IAC code. 2. Reload the system. 3. If the device that would cause the interrupt is not in the system, ignore the message.
K0D lprog rout reinc LEV.2	In program lprog is an invalid reference to subroutine rout at relative location reinc; i. e. , a LIBF reference to a type 4 or type 6 subroutine when it should be a CALL, or a CALL reference to a type 3 or type 5 subroutine when it should be a LIBF.	Correct the reference.

Table A-6. Skeleton Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
K0E lprog LEV. 2	The integer size or real precision indicated for program lprog does not agree with that indicated for the skeleton.	Correct the precision.
K13 rout LEV. 1, 2	The LEV. 1 message indicates that core load rout has been referenced via a CALL CHAIN, CALL SPECL, CALL QUEUE, CALL QIFON, or CALL UNQ, but rout is not in FLET.	Build the core load (*STORECI).
	The LEV. 2 message indicates that a required system area is not present.	Define the required area (*DEFINE CONFIG).
K14 lprog reinc LEV. 2	Program lprog contains an INSKEL COMMON reference at relative location reinc that falls outside the bounds of INSKEL COMMON.	Correct the reference in lprog.

Table A-6. Skeleton Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
K18 LEV.1	During a skeleton rebuild operation, the total length or the length of entries of the new ICL table are incompatible with those of the old ICL table.	Use same TASK and System Director as used in previous skeleton build operation.
K19 LEV.2	The number of interrupt levels defined for TASK is different from that defined for the System Director, or the length of the skeleton defined for the System Director is different from that defined on the *DEFINE CONFIG card.	<ol style="list-style-type: none"> 1. Correct definitions for number of interrupt levels. 2. Correct definition for length of skeleton.

Table A-7. System Loader Error Messages

Error Code	Meaning	Recovery
L01	A control card is missing.	Examine the system deck and prepare the necessary card. (See section 7.) Then place that card and all cards that should follow it in the card reader input hopper. Press reader START and console STEP.

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L02	An assignment card contains a nonnumeric value for an interrupt level, bit, IAC, or LUN.	Correct the error; replace the corrected assignment card in the system deck; place the *ASSIGNMENT card and the entire deck of assignment cards in the card reader input hopper. Press reader START and console STEP.
L03	The total bits specified on the assignment card does not agree with the count specified in columns 4 and 5 of the card.	Same procedure as for L02.
L04	A continuation card is defined for a different level than the assignment card it continues.	Same procedure as for L02.
L05	One of three conditions exists: <ol style="list-style-type: none"> 1. The interrupt level specified on the assignment card is greater than 23, but is not 99. 2. More than 16 bits are specified on an assignment card. 3. An IAC or LUN greater than 64 is specified on an assignment card. 	Same procedure as for L02.

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L06	An assignment card entry for IAC or LUN is not followed by a slash, comma, or blank.	Same procedure as for L02.
L07	A duplicate IAC code has been specified on an assignment card.	Same procedure as for L02.
L08	An attempt has been made to assign a LUN to an IAC that has no FORTRAN ID and, thus, no LUN.	Same procedure as for L02.
L09	A LUN equal to zero or to a value greater than 44 has been specified on an assignment card.	Same procedure as for L02.
L10	The same LUN has been assigned to more than one IAC.	Same procedure as for L02.
L11	More than one process interrupt has been assigned to a single interrupt level.	Same procedure as for L02.
L12	Duplicate interrupt level assignments have been made.	Same procedure as for L02.

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L13	Program name from *LDDSK card was not found in LET.	Verify proper format and spelling on *LDDSK card. If the card is correct, reload the LET deck, preceded by the LDDSK. LET card. This deck should be followed by the *LDDSK. name subroutines. (Note: this action clears all previous LET entries.) If the *LDDSK card is incorrect, correct the error, place the corrected card and the proper absolute program in the card reader input hopper, and press reader START.
L14	The specified relocatable subroutine already has a LET entry. The LET entry may be cleared by performing a DUP DELETE function.	The subroutine cannot be loaded. Bypass this subroutine and continue with the following control card or program deck.
L15	Checksum error.	1. To initiate a retry following this error, place the binary card in the card reader input hopper and press reader START. If the error persists, enter a punch in row 9 of column 3 to override the checksum.

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L16	The last program deck before the *CCEND card did not contain an end-of-program card (type F).	<p>2. To abort the loading of the program that caused the error, bypass the remaining records and continue with the next program or control card in the system deck.</p> <p>1. Correct the error. Then:</p> <ul style="list-style-type: none"> ● For an absolute program, place the entire program deck, preceded by the appropriate *LDDSK card, in the card reader input hopper and press reader START and console STEP. ● For a relocatable program, place the entire program deck in the card reader input hopper and press reader START and console STEP. <p>2. To bypass the program deck, skip to the next header or control card, place the rest of the cards in the card reader input hopper, and press reader START and console STEP.</p>

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L17	Either a type A card is out of order in an absolute deck or the header card is missing from the deck.	Place the proper *LDDSK card, followed by the corrected program deck, in the card reader input hopper and press reader START and console STEP.
L18	Illegal back origin encountered.	<p>1. To recover: reassemble, place the *LDDSK card and reassembled program deck in the card reader input hopper, and press reader START and console STEP.</p> <p>2. To continue without corrective action: Bypass the remaining cards of the program, place the remaining cards to be loaded in the card reader input hopper, and press reader START and console STEP.</p>
L19	*DEDIT card error. Either an invalid core size (must be 08, 16, or 32) was specified or too great a buffer size was specified (maximum 199).	Correct the *DEDIT card, place it in the card reader input hopper, and press reader START and console STEP.

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L20	Either there was no LET on disk or the *DEDIT card was read before LET was loaded on disk.	Place the *LDDSK.LET card, a valid LET deck, a *LDDSK.subr card, and the *DEDIT card in the card reader input hopper. Press reader START and console STEP.
L21	Either there is an illegal card type in the program deck or the header card or type F card is missing from the subroutine deck.	Same procedure as for L16.
L22	IOU subroutine is missing.	Place the *LDDSK.subr card followed by the IOU subroutine deck and the *DEDIT card in the card reader input hopper and press reader START and console STEP.
L23	DCOM not found on disk.	Place the *LDDSK.DCOM card, the DCOM deck, and the *DEDIT card in the card reader input hopper. Press reader START and console STEP.

Table A-7. System Loader Error Messages (Cont.)

Error Code	Meaning	Recovery
L24	The cards in the program deck preceding the *LDDSK card were not in the proper sequence.	Ensure proper sequencing of program deck: header card, data card, and EOP card. To continue, follow same procedure as for L16.
L25	An IAC code has not been assigned to the timer, disk, card reader, card punch, line printer, or console interrupt.	Same procedure as for L02.
L26	Either there is a sequence number error on a type A binary card or a type A card is missing or out of sequence.	Same procedure as for L16.

A.7 NONPROCESS MONITOR SUPERVISOR ERROR MESSAGES

The Supervisor error messages are identified by a three-character symbol, Nxx (N specifies the Supervisor messages and xx is a two-digit sequence number). These messages are listed in table A-8.

Table A-8. Supervisor Error Messages

Error Message	Meaning
N01 ILLEGAL MTR CD	A card has a slash in column 1 but does not contain a mnemonic acceptable to the Nonprocess Monitor Supervisor (e.g., //DUMP). The no execute indicator (INOEX) is set. The job will be assembled and stored but will not be executed.
N02 LDR CD ERROR	A loader card violates the specifications for its card type. The erroneous card is printed immediately following the N02 message. The no load and the no execute indicators are set. The job is assembled and stored, but it will not be loaded as part of the core load nor will it be executed.
N03 NOT CONTROL CD	The Supervisor has just encountered a card that it cannot identify; i.e., column 1 does not contain a slash (/) or an asterisk (*). The card image is printed immediately after the N03 message, and cards are bypassed until the next control card is read.
N04 READY READER	The Supervisor could not read from the card reader. After printing this message, the Supervisor loops until the reader is not busy and ready.
N05 NAME ERROR	A //FOR, //ASM, or //XEQ control card contains a name that is not in a valid format. The erroneous card is printed immediately following the N05 message. The no execute indicator is set. If the card is a //FOR or //ASM, the no load indicator is also set.

Table A-8. Supervisor Error Messages (Cont.)

Error Message	Meaning
N06 LDR CD SEQUENCE	The Supervisor has encountered a valid but unnecessary loader control card (e.g., an *INCLD without a preceding //XEQ), or else Nonprocess Monitor control records contain a control card that is applicable only to process jobs (e.g., an *RCORD following a //XEQ).
N07 LABEL FORMAT	The JOB card contains an erroneous pack label specification field. This error can be the result of trailing blanks or an alphabetic character in a numeric field. The job is aborted.
N08 ILLEGAL LDR CD	A card has an asterisk in column 1 but does not contain a mnemonic acceptable to the Nonprocess Monitor Supervisor (e.g., *ASM).
N09 NO LOAD	The no-load indicator has been set, and a store store core image operation (*STORECI) is requested. Although the loader is called, the store operation will not be performed.
N10 NO EXEC	The no-execute indicator (INOEX) has been set, and program execution is requested. Although the loader is called, the program will not be executed.
N11 LABEL ERR DRx	The label that appears on disk pack x does not match the label specified for that pack on the JOB card. The job is aborted.

Table A-8. Supervisor Error Messages (Cont.)

Error Message	Meaning
N12 TEMP DR ERR	Either the temporary drive is not in use or the drive number selection is not valid for the system. The job is aborted.
N13 FLET ERR DRx	The FLET entry for the disk pack on drive x contains a drive number that does not match the drive on which the pack is mounted. The job is aborted.
N14 NO LET/FLET DRx	Either drive x has no LET/FLET table or the table is not properly constructed. The job is aborted.
N15 *CCEND MISSING	A monitor control record has been encountered before processing of loader control records was completed (i. e. , before an *CCEND card was read). The loader will not be called; therefore, loading or execution will not occur. The monitor control record will be processed.
N16 PGM NOT IN FLET	The execution of a core image program has been requested via an XEQ card; however, the FLET table of the disk pack on the specified drive contains no entry for that program. If no drive is indicated on the card, all drives are searched.

Table A-8. Supervisor Error Messages (Cont.)

Error Message	Meaning
N17 NOT NP CORE LD	The execution of a core image program has been requested via an XEQ card. Although a FLET entry has been located, the program is not a nonprocess core load. The no load and no execute indicators are set, and the Supervisor reads the next control card.
N18 [ASM/FOR] NOT ON DISK	Although the specified program (ASM is Assembler; FOR is FORTRAN) has been removed from the system disk through a DUP DEFINE REMOV function, that program has been called by a monitor control record. The erroneous control record is printed immediately following the N18 message. The no execute indicator is set, and the Supervisor reads the next control card.

A.8 CORE LOAD BUILDER ERROR MESSAGES

The format of error messages produced by the Core Load Builder is the same as that for Skeleton Builder error messages:

Rxx lprog rout reinc LEV.y

where

- R identifies Core Load Builder message.
- xx sequence number; $00 \leq xx \leq 1F$.
- lprog name of the program being relocated.
- rout routine specified within lprog.
- reinc relocation increment of rout from lprog.

LEV.y denotes severity level of the error:

y = 0 - minor error/warning. Disk loading and/or execution is not suppressed.

y = 1 - moderate error. Although disk loading proceeds, execution is suppressed if the operation was initiated by a //XEQ control record. Core loads built with a level 1 error may be executed via *STORECI and //XEQ FX procedures.

y = 2 - severe error. Disk loading and execution are suppressed. If specified, the core load map will be printed. A level 2 error indicates a problem in a subroutine.

y = 3 - abort. Core Load Builder terminates its operations immediately and transfers control to the Supervisor or to DUP. A level 3 error indicates a problem in a mainline program.

The Core Load Builder error messages are listed in table A-9.

Table A-9. Core Load Builder Error Messages

Error Message	Meaning	Recovery
R01 LEV.3	A load table overflow has occurred. This situation causes immediate termination of Core Load Builder operation.	<ol style="list-style-type: none">1. As appropriate, decrease number of sub-routines, files, and/or in-core interrupt routines.2. Perform core load build operation off-line under TASK.

Table A-9. Core Load Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
R02 lprog LEV. 2, 3	A disk format error has been found in program lprog.	Correct the program on disk.
R03 rout LEV. 2	An entry for relocatable program rout cannot be found in LET on disk.	<ol style="list-style-type: none"> 1. Delete the reference to rout from the core load. 2. Assemble or compile program rout and enter it in LET.
R04 lprog LEV. 2	A subroutine that is specified as a LOCAL is not a type 3 or type 4.	Delete the subroutine name from the LOCAL control card and rebuild the core load.
R05 lprog rout reinc LEV. 0, 1, 2	<p>Within program lprog the call to TSS subroutine rout is invalid for the type of core load being built.</p> <p>The CALL statement is at relative location reinc of lprog.</p>	Correct the CALL statement.
R06 lprog rout reinc LEV. 2	The required core load name in the list of the TSS program sequence change CALL to rout at relative location reinc of lprog does not occur in an EXTERNAL statement in program lprog.	Correct lprog.

Table A-9. Core Load Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
R07 lprog llbb LEV. 2	The in-core interrupt subroutine lprog has been designated to service llbb, which has previously been assigned to another in-core interrupt subroutine.	Correct the assignment of in-core interrupts.
R08 LEV. 2	The disk drive that was designated for temporary system usage does not contain enough nonprocess working storage space to build the core load.	<ol style="list-style-type: none"> 1. Change the temporary drive to another drive. 2. Delete programs from the user's area to provide sufficient nonprocess working space.
R09 LEV. 2	The maximum length of 255 words (85 entries) for the transfer vector (ETV) has been exceeded.	Decrease the number of LIBF subroutines included within the core load.
R0A lprog LEV. 2	Program lprog has a back-origin that would overlay part of the skeleton or core load tables.	Correct the lprog back-origin.
R0B novfl LEV. 2	The sum of the lengths of the core load and COMMON exceeds the length of variable core.	Reduce the size of the core load and/or COMMON by the number of words specified in the message by novfl.

Table A-9. Core Load Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
R0C lprog iaccd LEV.0	The interrupt service sub-routine lprog contains an interrupt service entry point, defined for IAC code (iaccd), which is not in the system master branch table. The IAC number (iaccd) is hexadecimal.	<ol style="list-style-type: none"> 1. Correct the IAC code. 2. Reload the system. 3. If the device that would cause the interrupt is not in the system, ignore the message.
R0D lprog rout reinc LEV.2	In program lprog is an invalid reference to sub-routine rout at relative location reinc; e.g., a LIBL or CALL reference to a core load when it should be PNT.	Correct the reference.
R0E lprog LEV.2	The relocatable program lprog was not assembled or compiled within the TSS system.	Using the TSS system, assemble or compile program lprog.
R0F lprog LEV.1	The integer size or real precision indicated for program lprog does not agree with that indicated for previously loaded programs in the core load.	Correct the size or precision.

Table A-9. Core Load Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
R10 drvno secno LEV. 1	The accumulated length of DEFINED FILES exceeds (by secno) the amount of working storage (either nonprocess or process working storage) currently available on drive drvno.	<ol style="list-style-type: none"> 1. Redefine file lengths. 2. Delete programs from the user area to provide sufficient working storage.
R11 filn6 recno LEV. 0	In order not to exceed the area of the equated data file (from *FILES control record), the define file number (filno) and the number of records (recno) have been truncated.	<ol style="list-style-type: none"> 1. Enlarge the equate file. 2. Adjust the *DEFINE FILES statement in the main program.
R12 LEV. 3	The nonprocess system does not allow building of an interrupt, combination, or process-mainline core load.	Execute skeleton build procedure before building the core load.
R13 rout LEV. 2	The required core image data file rout cannot be found in FLET on disk.	Use DUP to allocate the named core image area on disk.

Table A-9. Core Load Builder Error Messages (Cont.)

Error Message	Meaning	Recovery
R14 lprog LEV.2	The INSKEL area defined for the system has been exceeded by the amount of INSKEL COMMON used by program lprog.	<ol style="list-style-type: none"> 1. Reduce the amount of INSKEL COMMON used by lprog. 2. Redefine the size of INSKEL area. This procedure requires rebuilding the system and all previously built core loads.
R15 lprog LINK LEV.2	Program lprog, built under the XEQ function, contains a CALL LINK statement, which is invalid.	Use the *STORECI function to build all links of the chain.
R16 lprog rout LEV.3	A LOCAL subroutine (rout) called another subroutine designated as a LOCAL in a different group.	<ol style="list-style-type: none"> 1. Change the calling sequence. 2. Change the designation of the LOCALs on the control cards.
RE1 LEV.3	An attempt was made to build a core load under an incomplete system.	Build an off-line or on-line system.

A.9 COLD START NAME CARD ERRORS

The error codes produced during the processing of a cold start name card are listed in table A-10.

Table A-10. Cold Start Name Card Error Codes

Error Code	Meaning
1	The cold start name is incorrect (*CLDST).
2	Illegal first character of initial core load name.
3	Column 7 is not blank.
4	Column 13 is not blank.
5	Column 15 is not blank.
6	Column 17 is not blank.
7	Column 19 is not blank.
8	Column 21 is not blank.
9	Column 23 is not blank.
10	Either an incorrect disk logical drive assignment was given or the drive is not on-line.
11	Either the core load name was not in FLET or the drive has not been assigned to the system.

To recover from error codes 1 through 9, follow this procedure:

1. Remove the cold start name card from the reader.
2. Correct the error.
3. Place the corrected name card in the reader.
4. Ready the reader.
5. Press console STEP.

To recover from errors 10 and 11, reload the cold start deck.

A.10 SYSTEM DIRECTOR EAC ERROR MESSAGES

The System Director EAC error messages have the format:

* cxx tt.ttt ac-m prog loc

where

* an asterisk indicates a process core load in core. A blank indicates a nonprocess core load in core.

c a code letter to designate the type of error:

F = FORTRAN
I = general input/output
M = mask
P = process input/output
Q = Queue
X = miscellaneous

xx a two-digit number denoting the classification of the error.

tt.ttt time, in thousandths of an hour.

ac-m the first two characters (ac) identify the area code for the associated I/O device; the third character (m) is a modifier and is applicable only where there is more than one device for that area code.

prog name of the program in core at the time the message is produced. This program may or may not be the one that originated the error condition.

loc location of the call leading to the error.

The System Director EAC error codes and their meanings are listed in table A-11.

Table A-11. System Director EAC Error Codes

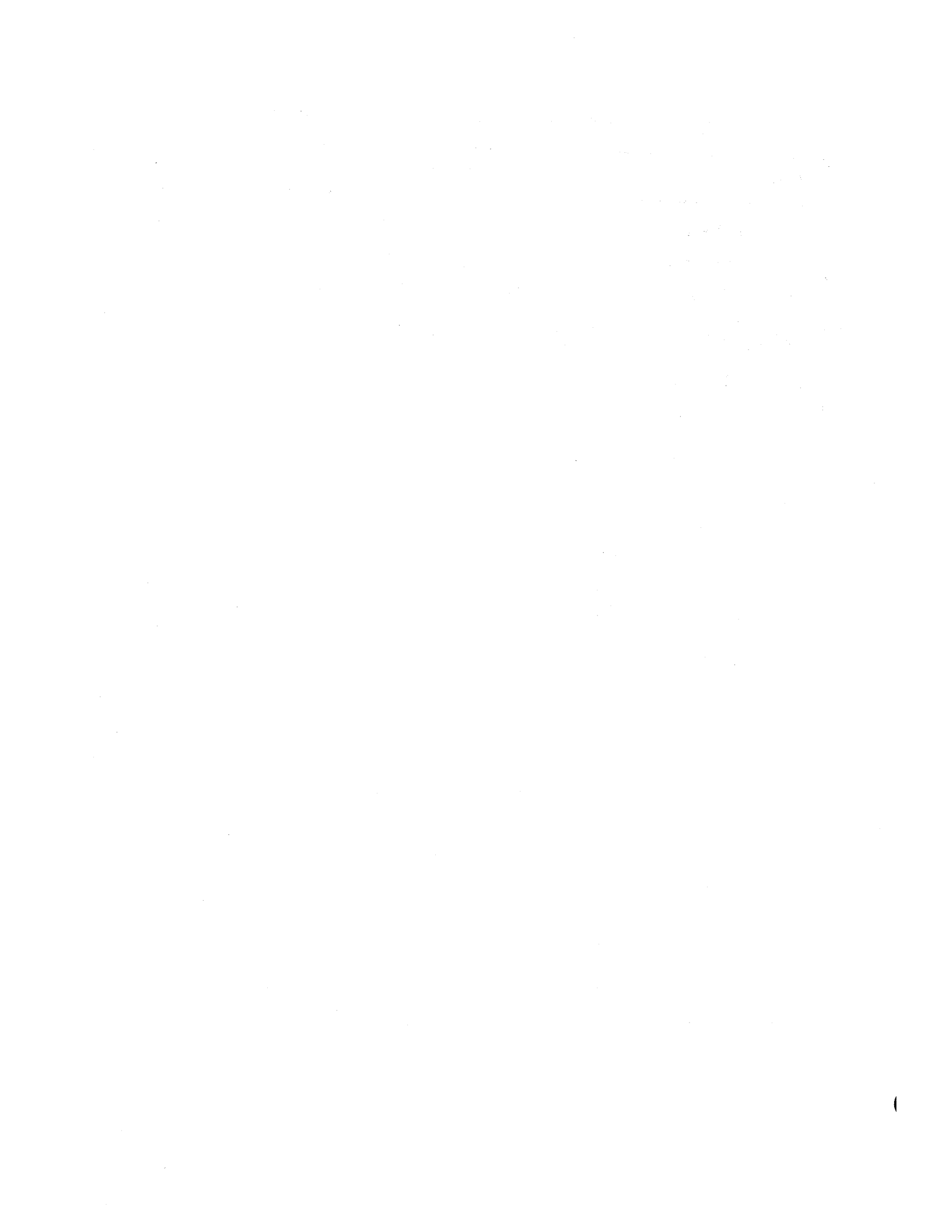
Code	Meaning
<u>INTERNAL ERRORS</u>	
MLTP EAC	An error has occurred while EAC was processing a previous error. Cold start required
996	Channel Address Register check error. User is given the option to RELOAD (if error is in skeleton), RESTART (if error is in variable core, abort nonprocess job; or execute user's restart core load for process job), or COLD START (required if EAC is unable to reload system).
997	Operation code violation. See code 996 for options.
998	Storage protect violation. See code 996 for options.
999	Parity error. See code 996 for options.
<u>USER ERROR TYPE CODES FOR DP I/O</u>	
I01	Parity error
I02	Storage protect violation
I03	Illegal CALL
I04	Not ready
I05	//blank card
I06	Feed check
I07	Read-punch check
I08	Data overrun
I09	Write select
I10	No print response
I11	Data error
I12	Invalid message on disk
I13	File protect error
I14	Tape error
I15	Excessive tape errors

Table A-11. System Director EAC Error Codes (Cont.)

Code	Meaning
I16	End of tape
I17	Invalid call to error routine
I18	No response from disk
I19	Invalid disk address
<u>USER ERROR TYPE CODES FOR PROCESS I/O</u>	
P01	Parity data or command reject
P02	Storage protect violation
P03	Illegal CALL
P04	Parity control
P05	Overlap conflict
<u>USER ERROR TYPE CODES FOR QUEUING</u>	
Q01	Error option is zero; call ignored
Q02	Error option is not zero; no lower priority in the queue
Q03	Queue entry replaced by new CALL QUEUE
<u>USER ERROR TYPE CODES FOR FORTRAN</u>	
F87	Illegal unit reference for unedited I/O
F88	Read list exceeds length of write list
F89	Record does not exist for read list element
F90	Illegal address computed in an indexed store operation
F91	Illegal integer variable used in a Computed GO TO statement
F92	Disk file not defined
F93	Disk record too large, zero, or negative
F94	Input record is in error
F95	Range of numerical values is in error
F96	Output field too small to contain the numbers

Table A-11. System Director EAC Error Codes (Cont.)

Code	Meaning
F97	Illegal unit reference for nondisk I/O
F98	Requested record exceeds allocated buffer
F99	Working storage area insufficient for defined files
<u>USER ERROR TYPE CODES FOR MASK ROUTINES</u>	
M01	Illegal CALL RESMK statement
M02	Illegal CALL UNMK statement
<u>USER ERROR TYPE CODES FOR PROGRAM SEQUENCE CONTROL</u>	
X01	Illegal CALL BACK statement
X02	Interrupt level error
X03	Core load not loaded on disk
X04	Restart core load not loaded on disk



APPENDIX B – CALLING SEQUENCES FOR SYSTEM ROUTINES

The routines described in this appendix are arranged in alphabetical order by mnemonic:

<u>Mnemonic</u>	<u>Paragraph</u>	<u>Mnemonic</u>	<u>Paragraph</u>
BACK	B.1	OPMON	B.13
CHAIN	B.2	QIFON	B.14
CLEAR	B.3	QUEUE	B.15
CLOCK	B.4	RESMK	B.16
COUNT	B.5	SAVMK	B.17
DPART	B.6	SETCL	B.18
ENDTS	B.7	SHARE	B.19
EXIT	B.8	SPECL	B.20
INTEX	B.9	TIMER	B.21
LEVEL	B.10	UNMK	B.22
LINK	B.11	UNQ	B.23
MASK	B.12	VIAQ	B.24

B.1 RESTORE SAVED CORE LOAD - BACK

Calling Sequence:

FORTRAN: CALL BACK

ASSEMBLER: CALL BACK

Operation: The BACK routine checks the indicator set by the SPECL routine (paragraph B.20) to determine if a core load has been saved in the special save area. If a mainline core load has not been saved, an error will result (see summary of system error messages in appendix A). If a core load has been saved, that core load is restored to variable core. Execution of the restored core load continues at the statement following the CALL SPECL statement.

B.2 EXECUTE PROCESS MAINLINE CORE LOAD - CHAIN

Calling Sequence:

```
FORTRAN:  CALL  CHAIN(NAME)
ASSEMBLER: CALL  CHAIN
           CALL  NAME
```

where

name - the name of a process mainline core load as it exists in FLET.

Operation: The call to the CHAIN routine terminates the present mainline core load. The named process mainline core load is loaded and executed. No reference to the process mainline queue is made.

B.3 CLEAR RECORDED INTERRUPTS - CLEAR

Calling Sequence:

```
FORTRAN:          CALL  CLEAR(M, L1, I1, L2, I2, . . . , Ln, In)
ASSEMBLER:        CALL  CLEAR
                  DC    M
                  DC    Li
                  DC    Ii
                  :
                  :
                  M    DC    value
                  Li  DC    value
                  Ii  DC    value
                  :
                  :
```

where

M - an integer value specifying the number of L and I parameters to follow; if M is zero, all indicators are cleared.

L - interrupt level or indicator.[†]

I - PISW bit position indicator or COUNT (paragraph B.5) indicators.[†]

[†]The parameters L and K for QIFON (paragraph B.14) for acceptable combinations of the L and I parameters.

Operation: The CLEAR routine enables the user to clear, selectively, the recorded interrupt indicators.

B.4 READ CLOCK - CLOCK

Calling Sequence:

FORTTRAN:	CALL	CLOCK(I)
ASSEMBLER:	CALL	CLOCK
	DC	I
	:	
	:	
I	DC	0

Operation: The CLOCK routine sets the integer expression referenced by the calling sequence to the current time. I is set to an integer value ($0 \leq I \leq 23999_{10}$), representing hours and thousandths of hours multiplied by 1000.

B.5 SET PROGRAM INTERVAL TIMER - COUNT

Calling Sequence:

FORTTRAN:	CALL	COUNT(S, T, N)
ASSEMBLER:	CALL	COUNT
	DC	S
	DC	T
	DC	N
	:	
	:	
S	DC	value
T	DC	value
N	DC	value

where

- S - number of the count subroutine to be executed when the specified time elapses; $0 \leq S \leq 31$.
- T - program interval timer to be set; $1 \leq T \leq 9$.
- N - number of intervals to which T is set.

Operation: The COUNT routine is identical to the TIMER routine except that COUNT uses software timers rather than hardware timers.

B.6 END COMBINATION CORE LOAD - DPART

Calling Sequence:

FORTRAN: CALL DPART
ASSEMBLER: CALL DPART

Operation: The CALL DPART statement is used as the last logical statement of a combination core load. The DPART routine checks the system level indicator to determine if the combination core load that called it is presently executing as a process mainline or interrupt core load. If the level indicator specifies that the mainline level is currently executing, the CALL DPART statement is executed as a CALL VIAQ statement (paragraph B.24). If an interrupt level is specified by the level indicator, the CALL DPART statement is executed as a CALL INTEX statement (paragraph B.9).

B.7 END TIME-SHARE OPERATION - ENDTS

Calling Sequence:

FORTRAN: CALL ENDTS
ASSEMBLER: CALL ENDTS

Operation: This routine is used by interrupt programs to set the time-share timer to zero. A CALL ENDTS statement has no effect unless a time-sharing operation is currently underway.

Example: An INSHEL interrupt servicing subroutine (ISS) senses a process condition to be serviced by a process mainline core load named CL009. Part of the ISS coding might be as follows:

```

:
CALL  QUEUE(CL009, 1, 0)  (see paragraph B.15)
CALL  ENDTS
CALL  INTEX                (see paragraph B.9)
:

```

Time-sharing always continues until the time-share timer has been decremented to zero. If the ISS mentioned above had interrupted a time-sharing operation (assembly, compilation, or execution of a nonprocess core load), that operation would be terminated the next time Timer C was updated, since Timer C is used to update the time-share timer. The Nonprocess Monitor, recognizing that the time-share interval was over, would save the nonprocess job in the nonprocess save area of the disk.

The last process mainline core load is restored, and control is given to its VIAQ routine (paragraph B.24). The VIAQ routine finds core load CL009 in the queue, so that core load is brought into core and executed.

If the CALL ENDTS statement had not been used, the time-sharing operation would have continued for the remaining time in the time-share timer. The time-sharing operation will be automatically continued the next time a CALL VIAQ statement is executed and the queue is empty.

B.8 END NONPROCESS CORE LOAD - EXIT

Calling Sequence:

```

FORTRAN:  CALL  EXIT
ASSEMBLER: CALL  EXIT

```

Operation: The EXIT routine is referenced by nonprocess core loads. It terminates the execution of the present nonprocess core load and transfers control to the Nonprocess Monitor, which reads the next card in the nonprocess job stream.

B.9 EXIT TO MASTER INTERRUPT CONTROL (MIC) - INTEX

Calling Sequence:

```
FORTTRAN:      CALL    INTEX
ASSEMBLER:     CALL    INTEX
```

Operation: The INTEX routine returns control to MIC on interrupt exit. It can be used only in interrupt core loads or INSKEL interrupt subroutines.

B.10 INITIATE PROGRAMMED INTERRUPT - LEVEL

Calling Sequence:

```
FORTTRAN:      CALL    LEVEL(I)
ASSEMBLER:     CALL    LEVEL
                DC      I
                :
                :
                I    DC    value
```

where

I - an integer expression specifying an interrupt level;
 $0 \leq I \leq 23$.

Operation: The LEVEL routine initiates a programmed interrupt on the interrupt level specified by the value of the integer (I) designated in the calling sequence.

The interrupt will be honored as soon as these two conditions exist:

- The requested level is unmasked.
- All higher priority levels have been serviced.

B.11 EXECUTE A NONPROCESS CORE LOAD - LINK

Calling Sequence:

```
FORTTRAN:      CALL    LINK(NAME)
ASSEMBLER:     CALL    LINK
                CALL    NAME
```


where

NAME - the name of a nonprocess core load in the core load (FLET) area. If the FORTRAN calling sequence is used, **NAME** must appear in an EXTERNAL statement.

Operation: A call to the LINK routine terminates the present nonprocess core load. The nonprocess core load named in the calling sequence (**NAME**) is loaded and executed. That nonprocess core load must have been built to reside in the core load (FLET) area prior to the execution of a CALL LINK statement.

B.12 SET SYSTEM MASK REGISTER - MASK

Calling Sequence:

FORTRAN:	CALL	MASK(MSK1, MSK2)
ASSEMBLER:	CALL	MASK
	DC	MSK1
	DC	MSK2
	:	
	:	
MSK1	DC	/xxxx
MSK2	DC	/xxxx

where

MSK1 } - two integer variables specify the bit pattern used to set the
MSK2 } system mask register. Bits 0 through 13 of MSK1 are used to set the mask status of interrupt levels 0 through 13. Bits 0 through 9 of MSK2 are used to set the mask status of interrupt levels 14 through 23. Bits 14 and 15 of MSK1 and bits 10 through 15 of MSK2 are not used.

Operation: The MASK routine uses the bit pattern of the two specified integer variables to set the system mask register. If the bit corresponding to any level is set (i. e. , is the digit 1), all interrupts associated with that level are inhibited. If the bit corresponding to any level is not set, the status of that level is not affected. Both parameters are required, even when the system is equipped with fewer than 14 levels.

Example: In the following coding sequence the call to the MASK routine will mask levels 1 through 3, 5 through 13, and 18 through 23. The mask status of levels 0, 4, and 14 through 17 will not be changed.

```
      :  
      :  
      CALL    MASK  
      DC      H77FF  
      DC      H0FFF  
      :  
      :  
H77FF  DC      /77FF    Mask Pattern: 0111 0111 1111 1111  
H0FFF  DC      /0FFF    Mask Pattern: 0000 1111 1111 1111  
      :  
      :
```

If the system were equipped with only 6 levels (i. e. , 0 through 5), the bits in the mask words corresponding to levels 6 through 23 would have no effect. However, the second mask word would still be required by the calling sequence.

B.13 RESET OPERATION MONITOR - OPMON

Calling Sequence:

```
FORTTRAN:      CALL    OPMON  
ASSEMBLER:     CALL    OPMON
```

Operation: The OPMON routine is used to reset the Operations Monitor (stall alarm).

B.14 QUEUE IF INDICATOR ON - QIFON

Calling Sequence:

```

FORTRAN:      CALL  QIFON(NAME, I, L, K, J)
ASSEMBLER:    CALL  QIFON
              CALL  NAME
              DC    I
              DC    L
              DC    K
              DC    J
              :
              :
I             DC    value
J             DC    value
L             DC    value
K             DC    value
    
```

where

NAME - the name of a process mainline core load, as it exists in FLET, to be placed into the queue.

I - the priority with which the named core load is to be placed in the queue; $1 \leq I \leq 32767$.

L, K - interrupt reference combinations:

<u>L</u>	<u>K</u>	<u>Meaning</u>
0 - 23	0 - 15	Level and bit; process interrupts
0 - 23	Any negative number	Programmed interrupts
Any negative number	0 - 31	Programmed timer (count) subprogram number

J - error parameter; same as parameter J for CALL QUEUE statement (see paragraph B.15); $0 \leq J \leq 32767$.

Operation: The QIFON routine tests a recorded interrupt indicator, as determined by the L and K parameters in the calling sequence. A recorded interrupt indicator is set when a process interrupt, a programmed interrupt, or a

programmed timer interrupt occurs, and the required servicing subroutine is not in core at the time the interrupt is received. This procedure does not apply to interrupts that have associated interrupt core loads assigned to them. If the indicator specified by the L and K parameters is set, the process mainline core load named in the calling sequence is placed in the queue as a delayed servicing mechanism. The L and K parameters are interpreted as follows:

1. If both parameters are positive, L indicates the level, and K indicates the bit of an external process interrupt.
2. If K is negative, L indicates the level of a programmed interrupt.
3. If L is negative, K indicates the number of a subroutine as specified in a CALL COUNT statement (paragraph B.5) which was not in core when the programmed timer interval elapsed.

Requirements: The NAME, I, and J parameters for the CALL QIFON statement are identical to the NAME, I, and J parameters for a CALL QUEUE statement (paragraph B.15) and must conform to the same restrictions.

B.15 QUEUE A CORE LOAD - QUEUE

Calling Sequence:

FORTRAN:	CALL	QUEUE(NAME, I, J)	
ASSEMBLER:	CALL	QUEUE	
	CALL	NAME	
	DC	I	
	DC	J	
	:		
	:		
	I	DC	value (priority)
	J	DC	value (error option)

where

NAME - the name of the process mainline core load to be placed in the process mainline queue.

- I - the execution priority to be associated with the queue entry;
 $1 \leq I \leq 32767$ (highest priority is 1; lowest priority is 32767).
- J - error option if queue is full:
- | | |
|-----------------------|--|
| J = 0 | If queue is full, output error message to EAC printer; then ignore call. |
| $1 \leq J \leq 32766$ | If queue is full, replace the lowest priority entry with this entry. Do not replace any entry with a priority number less than the value of J. If no replaceable entry can be found, execute a RESTART error recovery procedure. |
| J = 32767 | If queue is full, execute a RESTART error recovery procedure. |

Operation: The QUEUE routine places an entry into the process mainline queue, based on the core load named in the calling sequence. The same core load cannot be entered into the queue twice with the same priority. The second call is ignored. The same core load can be placed into the queue with different priorities. If two different core loads are entered with the same priority, the VIAQ routine (paragraph B.24) will execute the core load that was placed in the queue first.

Requirements: It is the user's responsibility to ascertain that the core load named in the calling sequence is a process mainline core load and that the core load has been built to reside in the FLET area before the CALL QUEUE statement is executed.

When applicable, the RESTART procedure will consist of executing the restart core load associated with the process mainline core load which is currently in VCORE or which last occupied VCORE.

If the FORTRAN calling sequence to the QUEUE routine is used, the name of the core load to be placed in the queue must appear in an EXTERNAL statement for the program.

B.16 RESTORE SYSTEM MASK REGISTER - REMSK

Calling Sequence:

FORTRAN:	CALL	REMSK(I, J)
ASSEMBLER:	CALL	REMSK
	DC	I
	DC	J
	:	
	:	
I	DC	/xxxx
J	DC	/xxxx

where

I	}	- two integer variables that specify the status levels to be masked. A 1 in a bit position causes the corresponding level to be masked; a 0 causes the corresponding level to be unmasked.
J		

Operation: The REMSK routine is normally used with the SAVMK routine (paragraph B.17). By referencing the same parameters, the REMSK routine will set the system mask to the state detected by SAVMK. REMSK uses bits 0 through 13 of the first parameter (I) to set the mask status of levels 0 through 13 and bits 0 through 9 of the second parameter (J) to set the mask status of levels 14 through 23.

B.17 SAVE SYSTEM MASK REGISTER - SAVMK

Calling Sequence:

FORTRAN:	CALL	SAVMK(I, J)
ASSEMBLER:	CALL	SAVMK
	DC	I
	DC	J
	:	
	:	
I	DC	0
J	DC	0

Operation: The SAVMK routine records the status of interrupt levels 0 through 13 in bits 0 through 13 of the first parameter (I) and the status of levels 14 through 23 in bits 0 through 9 of the second parameter (J). After the routine has been executed, a 1 in a bit position indicates that the corresponding level is currently masked; a 0 indicates that the corresponding level is currently unmasked.

B.18 SET SYSTEM REAL-TIME CLOCK - SETCL

Calling Sequence:

FORTRAN	CALL	SETCL(I)
ASSEMBLER:	CALL	SETCL
	DC	I
	:	
	:	
I	DC	value

where

I - any integer value in the range $0 \leq I \leq 23999_{10}$.

Operation: The SETCL routine sets the system real-time clock to the value of the variable specified in the calling sequence. This parameter specifies the time in hours and thousandths of hours, multiplied by 1000; for example, 6:00 a.m. would be expressed as 06000, or 11:30 p.m. as 23500.

B.19 INITIATE TIME-SHARING OPERATION - SHARE

Calling Sequence:

```
FORTRAN:      CALL    SHARE(I)
ASSEMBLER:    CALL    SHARE
              DC      I
              :
              :
              I    DC    value
```

where

I - an integer expression designating the number of time intervals to be allowed for nonprocess program operation.

Operation: The SHARE routine is used by process mainline core loads to initiate a time-sharing operation for a specific time interval. The value of I determines the duration of the time-sharing interval in units of the programmed timer base. Execution of the CALL SHARE statement suspends the execution of the mainline core load which called it until the interval has elapsed or until a CALL ENDTS statement (paragraph B.7) is executed from an interrupt program.

B.20 SPECIAL LINKAGE - SPECL

Calling Sequence:

```
FORTRAN:      CALL    SPECL(NAME)
ASSEMBLER     CALL    SPECL
              CALL    NAME
```

where

NAME - the name of the process mainline core load that is to be executed via the special linkage routine.

Operation: A call to the SPECL routine initiates the following sequence of operations:

1. All variable core is saved in the special save area on the disk (a special save area must have been designated at system generation time).

2. An indicator is set to inform the Skeleton Executive that a core load has been placed in the special area.
3. The core load named in the calling sequence is loaded and executed.

Requirements: The core load named in the calling sequence must have been built to reside in the core load (FLET) area. If the FORTRAN calling sequence is used, NAME must appear in an EXTERNAL statement.

B.21 SET INTERVAL TIMER - TIMER

Calling Sequence:

FORTRAN	CALL	TIMER(NAME, I, T)
ASSEMBLER:	CALL	TIMER
	DC	NAME
	DC	I
	DC	T
	:	
	:	
	NAME DC	address of subroutine NAME
	I DC	value
	T DC	value

where

- NAME - identifies the program to be executed after the specified time interval expires.
- I - identifies the interval timer: (1 or 2).
 - 1 = timer A
 - 2 = timer B
- T - specifies the number of intervals to which I is set (i. e., the number of timer intervals to elapse before execution).

Operation: The TIMER routine is used to obtain extremely accurate delays in the execution of a programming sequence. It should be used only for short time intervals because it is on a high level. Subroutine NAME must be in core when the timer interrupt is recognized.

B.22 UNMASK INTERRUPT LEVELS - UNMK

Calling Sequence:

```
FORTRAN:      CALL    UNMK(UNMK1, UNMK2)
ASSEMBLER:    CALL    UNMK
              DC      UNMK1
              DC      UNMK2
              :
              :
UNMK1 DC      /xxxx
UNMK2 DC      /xxxx
```

where

UNMK1 } - two integer variables specify the bit pattern used to unmask
UNMK2 } the system interrupt levels. Bits 0 through 13 of UNMK1
correspond to levels 0 through 13; bits 0 through 9 of
UNMK2 correspond to levels 14 through 23.

Operation: The UNMK routine allows interrupts on the levels specified by the two variables designated in the calling sequence. A 1 in any bit position causes the corresponding level to be unmasked; a 0 bit allows the corresponding level to remain unchanged. Both parameters are required, even when the system is equipped with fewer than 14 interrupt levels.

Example: In the following coding sequence the MASK statement (paragraph B.12) will mask levels 1 through 3, 5 through 13, and 18 through 23. The UNMK statement will unmask levels 1, 2, 6 through 11, and 14 through 23.

```

      :
      :
      CALL  MASK
      DC    H77FF
      DC    H0FFF
      :
      :
      CALL  UNMK
      DC    UN1
      DC    UN2
      :
      :
H77FF  DC    /77FF      Mask Pattern: 0111 0111 1111 1111
H0FFF  DC    /0FFF      Mask Pattern: 0000 1111 1111 1111
UN1    DC    /63F0      Bit Pattern:   0110 0011 1111 0000
UN2    DC    /FFFF      Bit Pattern:   1111 1111 1111 1111
      :
      :

```

After these calls are executed, levels 0 and 4 would remain unchanged; levels 3, 5, 12, and 13 would be masked; and, all other levels would be unmasked.

If the system were equipped with only 6 levels (i.e., 0 through 5), the references in the calling sequences to levels 6 through 23 would have no effect. However, the second parameter would still be required by the calling sequence.

B.23 REMOVE CORE LOAD FROM QUEUE - UNQ

Calling Sequence:

```

FORTRAN:      CALL  UNQ(NAME, I)
ASSEMBLER:    CALL  UNQ
               CALL  NAME
               DC    I
               :
               :
I             DC    value

```

where

- NAME - the name of the core load to be removed from the queue.
I - the execution priority used when the core load was entered into the queue; $1 \leq I \leq 32767$.

Operation: The named core load is removed from the queue. If the named core load is not in the queue or if it is not currently in the queue with the specified execution priority (I), the call is ignored.

Requirements: If the FORTRAN call is used, the name of the core load must appear in an EXTERNAL statement.

B.24 EXECUTE A QUEUED CORE LOAD - VIAQ

Calling Sequence:

FORTRAN: CALL VIAQ
ASSEMBLER: CALL VIAQ

Operation: The normal function of the VIAQ routine is to return control to the System Director from a process core load. VIAQ searches the queue table and initiates execution of the highest priority process mainline core load found. If the queue is empty, variable core is made available for time-sharing operation. If no time sharing is requested, the system will wait until an interrupt is recognized.

APPENDIX C - DIFFERENCES BETWEEN TSX AND TSS

C.1 CARD INPUT/OUTPUT

For TSS the CARDN routine must always be included in the skeleton. The routine has been written to double buffer all card input/output. Because of this approach, two "pusher" cards are required at the end of each deck fed into the card reader.

This approach also poses a small problem with the Absolute Loader and System Loader. When a card-out-of-sequence or checksum error occurs, two cards must be removed from the card reader stacker and placed back in the card reader hopper. After the console STEP switch has been pressed, another error message will be printed stating that the problem has not been corrected. This printout occurs because the next card had already been read and placed in CARDN's core buffer. Press STEP again to continue operation.

C.2 LINE PRINTER

The TSS line printer driver PRNTN is double buffered. Because of this buffering, the programmer need not be concerned about a buffer busy test after a call to PRNTN. However, if a program is being written to run under control of TSX, TSS, and DM2, the buffer busy test should be made.

C.3 DISK INPUT/OUTPUT

As far as the user is concerned, the only new feature with the disk is the ability to have up to ten platters on the system. The first three platters are the system-supported drives; additional drives may be addressed through DISKN. Since all disks are on a single data channel, simultaneous read/write operations are not allowed.

C.4 TYPEWRITER/KEYBOARD

Up to eight keyboards are allowed with TSS. With TSS read requests may be intermixed freely with write requests. Read requests will not be honored

until all pending write requests are completed. This prevents the keyboard from locking out the printer portion of the TTY prematurely.

C.5 SENSE SWITCHES

There are no sense switches on the GA 18/30 Computer. Therefore, any request for data through the console must be through the data switches. The IOCC for sensing the sense switches on the 1800 is the same as that used to sense the status of the console TTY on the GA 18/30.

C.6 CLEARING CORE

There is no CLEAR CORE feature on the GA 18/30 Computer as there is on the 1800. To enable the user to clear storage protect bits and set core to zero as required by TSS, a ZAP card is provided. This is a one-card program in IPL format. It is loaded via the IPL feature to core from the card reader at location zero. Execution begins at core location zero. After core has been cleared, the program comes to a WAIT.

C.7 RTR INSTRUCTIONS

The register transfer instructions on the GA 18/30 are not supported by the TSS assembler. To incorporate these instructions in a program, convert them to DC format.

GLOSSARY

The definitions given here are for the terms as they are used in this document.

A

absolute coding - Coding that uses instructions with absolute addresses.

Contrast with "relocatable coding."

ASM - See "Assembler."

Assembler - The assembler program that is included in the Nonprocess Monitor.

B

background processing - The automatic execution of lower priority computer programs when higher priority programs are not using the system resources.

Contrast with "foreground processing."

buffer - Intermediate storage area between two data processing storage or data handling systems with different access times or formats. An interim system to facilitate interface between two other systems.

C

cold start - The procedure of loading the Cold Start Program, whose function is to load the Skeleton Executive into core, storage protect it, start the real-time clock, and call the user's initial core load for execution. Thus, this procedure places the System Director in control of the on-line system.

combination core load - A core load that can be executed as either an interrupt or a mainline core load.

common areas - Three areas of core storage are used for FORTRAN COMMON storage: INSKELEL common, interrupt common, and core load common. See individual definitions.

core image format - The format in which core loads are stored on disk, which is the same form they have when they are in core storage to be executed (i. e., linkages provided, etc.).

core load - A complete, executable programming unit, which is stored in core-image format on disk. It consists of a main program (interrupt, mainline, or nonprocess), all required subroutines that are not permanently in core, and the communications areas. Further, mainline core loads may include in-core interrupt routines.

core load common - Located at the high-address end of core storage and referenced only by mainline or nonprocess core loads. See also "common areas."

D

data processing input/output - Refers to general input/output devices, such as printers, card readers/punches, teletypewriters, as compared with process input/output devices. Subroutines are supplied with the TSS system that enable the user to reference data processing I/O devices easily.

disk - See "magnetic disk storage."

Disk Utility Program (DUP) - A set of disk handlers (routines) included in the Nonprocess Monitor.

DUP - See "Disk Utility Program."

E

EAC - See "Error Alert Control Program."

Error Alert Control Program (EAC) - One of several programs that constitute the System Director.

exchange - A save operation followed immediately by the overlaying of VCORE with a new core load. Also referred to as "swapping."

F

feedback - In a control system, feedback is the signals fed back from a controllable process to denote its response to the command signal.

Fixed Location Equivalence Table (FLET) - Serves as a map for the location of core loads and data files. Each core load and data file requires at least one entry in FLET. See also "Location Equivalence Table."

FLET - See "Fixed Location Equivalence Table."

FOR - See "FORTRAN."

foreground processing - The automatic execution of the programs that have been designated to preempt the use of the computing facility; usually a "real-time" program. Contrast with "background processing."

FORTRAN - The compiler that is included in the Nonprocess Monitor.

G

generate - To produce a program by selection of subsets from a set of skeletal coding under control of parameters.

generator - A controlling routine that performs a generate function; e.g., core load builder, skeleton builder.

H

hardware - The equipment or "machinery" used in a computer system. The computer itself and peripheral devices as opposed to "software," which denotes written information.

I

INSKEL Common - Located within the system skeleton and can be referenced by any process or nonprocess program. See also "common areas."

interface - In control terminology, the means used to link components in a control system. In computer terminology, a common boundary between automatic data processing systems or parts of a single system.

interrupt - To stop a process in such a way that it can be resumed.

Interrupt Common - Located at the high-address end of the interrupt core load save area and used for interprogram communication between programs that form an interrupt core load or between combination core loads when they are executed on the mainline level. See also "common areas."

interrupt core load - A program unit that resides on disk and is brought into core to service a particular interrupt.

interrupt program - A program that is executed as the result of a particular interrupt. Same as interrupt routine and interrupt servicing routine.

Interrupt Status Table - Specifies which interrupt routines are in core with the current mainline core load and contains the entry address for the interrupt routine.

interval timer - A clocking device that cycles a value contained in a word of main storage, enabling the computer system to read elapsed time.

Interval Timer Control Program (ITC) - One of several programs that constitute the System Director.

ITC - See "Interval Timer Control Program."

J

job - A specified group of items prescribed as a unit of work for a computer. By extension, a job usually includes all necessary computer programs, linkages, files, and instructions to the operating system.

L

LET - See "Location Equivalence Table."

level work areas - Contain interrupt level instructions, MIC linkages, and work areas. One level work area is required for each interrupt level used, process mainline, nonprocess core load, and internal error level.

linkage - In programming, coding that connects two separately coded routines.

LOCAL subprograms - Subprograms that are read from disk into core for execution when called by the object program. All LOCALs associated with a program use the same area of core storage by overlapping each other as they are called.

Location Equivalence Table (LET) - Serves as a map for supplied and relocatable programs. Each relocatable program or subroutine stored on disk has at least one entry in the table. An entry contains the name of the item and location information. Each entry point in a subroutine requires an entry in LET. All operations that involve including or deleting relocatable programs reference LET. (Core loads and data files refer to FLET.) See also "Fixed Location Equivalence Table."

M

magnetic disk storage - A storage device or system consisting of magnetically coated disks, on the surface of which information is stored in the form of magnetic spots arranged in a manner to represent binary data.

Mainline Core Load Queue Table - See "Queue Table."

mainline program - A program that does not directly service an interrupt (i. e., analysis program); it executes on the lowest interrupt level.

map - To establish a correspondence between the elements of one set and those of another set. [noun:] The listing that represents this correspondence.

mask - A pattern of characters that is used to control the retention or elimination of portions of another pattern of characters.

Master Interrupt Control Program (MIC) - One of several programs that constitute the System Director.

MIC - See "Master Interrupt Control Program."

N

nonprocess core load - A core load that is executed under control of the Nonprocess Monitor.

Nonprocess Monitor - An independent programming system that operates under the System Director in a time-sharing environment or under TASK in an off-line situation to control the execution of supplied (i. e. , FORTRAN, Assembler, Disk Utility Routines) or user-written, nonprocess programs.

nonprocess program - A program that executes under control of the Nonprocess Monitor and does not (normally) perform any function related to the process under control. Such a program may be supplied (i. e. , compiler or assembler) or user-written (e. g. , payroll application).

Nonprocess Supervisor (SUP) - One of several programs included in the Nonprocess Monitor.

nonprocess work storage area - A temporary storage area on disk used during the execution of nonprocess programs and extensively during Nonprocess Monitor operations (e. g. , to store the object program as it is generated by the Assembler or compiler).

O

object code - Output from a compiler or assembler which is itself executable machine code or is suitable to produce executable machine code.

off-line - A mode of operation in which the GA 18/30 Industrial Supervisory System is not monitoring a process and, therefore, can be operated as any general purpose computer might be.

on-line - A mode of operation in which the GA 18/30 Industrial Supervisory System, utilizing the Time-Sharing Executive System (TSS), monitors the process under control.

P

peripheral equipment - Equipment that is external to and not part of the central processing instrumentation; includes such equipment as paper tape punches and readers, disk storage units, line printers, and typewriters.

process input/output - Refers to special input/output devices that are specific for an installation and the process to be controlled. Process input/output subroutines are not included in the supplied routines.

process program - A program, normally user-written, that performs some function in relation to controlling the process. There are two types of process programs: mainline and interrupt.

process work storage area - A temporary storage area on disk used during the execution of process programs.

Program Name Table - Consists of the name and disk address of any core loads that are called by the current core load and the name of a core load specified for restart.

Program Sequence Control Program (PSC) - One of several programs that constitute the System Director.

PSC - See "Program Sequence Control Program."

Q

Queue Table - The Mainline Core Load Queue Table contains the names of mainline core loads and their respective priorities that have been queued for future execution.

R

real time - Pertaining to the performance of a computation during the actual time that the related physical process transpires in order that results of the computation can be used in guiding the physical process.

reentrant program - One that can be interrupted at any point, employed by another user, and then resumed from the point of interruption. All supplied routines that are required on multiple levels in TSS are fully reentrant.

relocatable program - One that has been assembled or compiled but has not been converted into core loads.

relocate - In computer programming, to move a routine from one portion of storage to another and to adjust the necessary address references so that the routine, in its new location, can be executed.

restore - The operation by which the contents of a disk save area are returned to VCORE.

S

save - An operation by which all or a portion of the variable core area is moved to a save area on disk so that it will not be destroyed when a higher priority operation is read into VCORE.

Skeleton Executive - The basis, or framework, of an on-line TSS system; it must be resident in core. It consists of both supplied and user-written routines. Synonymous with system skeleton when referring to an on-line system.

software - Programs, routines, codes, and other written information used with digital computers as distinguished from "hardware," the equipment itself.

subroutine library - Includes the most frequently required subroutines used by the process and nonprocess programs.

SUP - See "Nonprocess Monitor."

System Director - That portion of the Skeleton Executive that handles all interrupts, controls user-specified sequence of process control programs, and controls the time-sharing of nonprocess programs.

system generation - The procedure of assembling, storing on disk, and preparing for execution all elements necessary to constitute a TSS system for the specific installation.

system skeleton - The permanently assigned area of core storage that contains the framework of the system, such as programs, work areas, communication areas, and user-defined options. See also "Skeleton Executive" and "Temporary Assembled Skeleton."

T

TASK - See "Temporary Assembled Skeleton."

Temporary Assembled Skeleton (TASK) - The executive system for off-line operation; it provides two services: it serves as the vehicle by which the TSS system is tailored for a specific installation, and it serves as the system skeleton for the Nonprocess Monitor's off-line operation.

time-sharing - The ability to use the computer to execute nonprocess programs during times when the process programs are not being executed. The Skeleton Executive retains the ability to respond to process interrupts.

Time-Sharing Control Program (TSC) - One of several programs that constitute the System Director.

TSC - See "Time-Sharing Control Program."

V

variable area (VCORE) - That area of core outside the system skeleton; it is used by process and nonprocess core loads and by TSS programs such as the Nonprocess Monitor.

VCORE - See "variable area."

W

work area - See "level work areas."

BIBLIOGRAPHY

GA 18/30 FORTRAN IV Reference Manual, publication no. 88A00123A

GA 18/30 Industrial Supervisory System, Programming/Operations Manual,
publication no. 88A00121A

GA 18/30 Industrial Supervisory System, Reference Manual, publication
no. 88A00026A

IBM 1130 Assembler Language (Form C26-5927)

IBM 1130/1800 Basic FORTRAN IV Language (Form C26-3715)

IBM 1800 Time-Sharing Executive System, Concepts and Techniques
(Form C26-3703)

IBM 1800 Time-Sharing Executive System, Operating Procedures (Form C26-3754)

IBM 1800 Time-Sharing Executive System, Subroutine Library (Form C26-3723)

Louden, Robert K., Programming the IBM 1130 and 1800, Englewood Cliffs,
New Jersey: Prentice-Hall, Inc., 1967