File Number 1800-36
Form C26-3703-0

IBM Systems Reference Library

IBM 1800 Time-Sharing Executive System Concepts and Techniques

The purpose of this publication is to describe the facilities provided by the IBM 1800 Time-Sharing Executive (TSX) System, and to explain the basic concepts and techniques underlying their use. It is intended as a reference and guide for customer systems personnel in the implementation of the TSX system.

The manual is divided into four sections. The first section serves as an overall introduction to the TSX system. The second and third sections describe the three main executive programs and discuss some of the important design considerations that bear on the use of standard TSX components. The final section provides selections of programming techniques covering a wide spectrum of TSX usage.

The general approach taken is to explain each concept as it is encountered, and, where possible, elucidate that concept by means of an example. Numerous sample problems are included to acquaint the programmer with recommended techniques of TSX programming. A detailed TSX Sample System is specially provided as a tutorial on all aspects of TSX design, usage and implementation.

PREFACE

This publication describes the facilities provided by the IBM 1800 Time-Sharing Executive System, and discusses the concepts and techniques underlying their use. It is intended as a reference and guide for customer systems personnel in the implementation of the TSX system.

The manual is written in four progressive sections where information in one section is sometimes necessarily related to information in another section. These comprise:

- Overview of the IBM 1800 Time-Sharing Executive System
- Functions of Executive Programs
- System Design Considerations
- Programming Techniques

The approach taken is to explain each concept as it is encountered. In some instances, a subject concept is necessarily included in a section prior to its definition later on in that section or a subsequent section. Sample problems are scattered throughout the text as illustrative examples designed to clarify concepts discussed and to familiarize the user with recommended techniques. They should not be construed as models.

The first section gives a rapid survey of the TSX system. It defines the executive system, its modes of operation and system requirements; discusses some of the basic TSX system concepts employed; and describes the various components of the system, and their inter-relationships to the total system.

The second section describes the three main executive programs (TASK, the System Director, and the Nonprocess Monitor) in terms of their functions and capabilities. Numerous examples are included as demonstration of sound programming practice and technique. Subjects discussed embrace: Program Scheduling, Handling of Interrupts, Use of Interval Timers, Use of Time-Sharing, Error Alert Control and Procedures, and Nonprocess Monitor Usage.

The third section discusses some of the important design considerations bearing on the use of standard TSX system components such as the System Loader, IBM Nonprocess System, Temporary Assembled Skeleton (TASK), and the System Director. Subjects discussed include: Assignment of Interrupt Levels and Restrictions, Level Work Areas, Disk System Configuration, and the System Skeleton.

The final section incorporates selections of programming techniques covering a wide spectrum of TSX uses. The purpose of this section is to aid the programmer, acquaint him with recommended techniques of TSX programming, and to help him to build on the fundamentals discussed in earlier sections of this manual. A detailed TSX Sample System at the end of the section touches on every facet of TSX design, use and implementation.

For details of TSX system generation procedures, System Loader assignment cards, TASK and System Director equate cards, and all Nonprocess Monitor control cards, the user is referred to IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

To derive maximum benefit from "Concepts and Techniques", the user should have a working knowledge of the following TSX support publications:

IBM 1800 Data Acquisition and Control System, Functional Characteristics, Form A26-5918

IBM 1800 Assembler Language, Form C26-5882

IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715

IBM 1800 Data Acquisition and Control System, Data Processing Input-Output Units, Form A26-5969

IBM 1130/1800 Plotter Subroutines, Form C26-3755

IBM 1800 Time-Sharing Executive System Subroutine Library, Form C26-3723

First Edition

This manual obsoletes and replaces the IBM 1800 Time-Sharing Executive System, Specifications, Form No. C26-5990. Concurrent with this release, the subroutine portion of the Specifications Manual (C26-5990) is replaced by the IBM 1800 Time-Sharing Executive System, Subroutine Library, Form No. C26-3723.

CONTENTS

ILLUSTRATIONS

Figures

Tables

Program Listings

INTRODUCTION

With few exceptions, real-time applications are distinguished from other applications by two chief characteristics: 1) some process or operation going on outside the computer system normally has a continuous need for on-line communication with the system, 2) there is a requirement for the computing system to keep pace with the associated process or operation. These characteristics of the application place some unique and stringent requirements on real-time processing systems for use in the real-time environment.

Recognizing the formidable programming task associated with a system of this scope, IBM has developed the 1800 Time-Sharing Executive System (TSX) which relieves the user of much of the required programming effort by freeing him to concentrate on the primary task of problem solution. TSX is a FORTRAN-oriented disk-resident operating system which permits the user to make optimum use of an IBM 1800 Data Acquisition and Control System (DACS) for its primary purpose, the control of processes and similar complex environments, as well as providing him with an effective off-line monitor system for data processing and scientific computation. TSX improves greatly the versatility of a Data Acquisition and Control System (DACS) computer by making it possible for background jobs to be processed when the real-time foreground task relinquishes control of the processor-controller. It is through time-sharing that idle computer time is minimized or eliminated. Programs may be written in FORTRAN and/or symbolic Assembler language.

Figure 1 introduces the capabilities of TSX in generalized form.

MINIMUM SYSTEM REQUIREMENTS

To assist users in performing their initial system generation, a standardized "starter" called System Generation (SYSGEN) TASK is provided with each installation which contains the basic elements necessary for system generation in a form that will be directly usable by a majority of users. SYSGEN TASK is supplied in assembled object format as part of the IBM Nonprocess System and consists of the following:

- Nonprocess Monitor Linkages
- Skeleton Builder Linkages
- Absolute Loader

It is designed to support the following minimum devices:

- 1 IBM 1801 or 1802 Processor-Controller with a minimum of 8K words of core storage
- 1 IBM 2310 Disk Storage Unit with one disk drive
- 1 IBM 1442 Card Read Punch
- 1 IBM 1816 Keyboard Printer (that is, printer portion only) or
- 1 IBM 1053 Printer

The user may employ additional I/O devices on his system, but he must satisfy the above machine configuration requirements to use SYSGEN TASK. For example, if he substitutes a 1443 Printer for a 1053 Printer or an 1816 Keyboard Printer, a card assembly of the TASK source deck to include this provision becomes mandatory. As the "starter" system is a limited version of the Temporary Assembled Skeleton (TASK), it will neither buffer 1053 Printer messages to disk, nor does it contain the trace and dump utility functions.

Machine Features Supported

In addition to the above, the following optional machine units and features are supported by the TSX system:

- Additional core storage (up to a maximum of 32,768 words)
- Additional disk drives for IBM 2310 Disk Storage Unit -- up to a maximum of three disk drives
- Additional IBM 1442 Card Read Punch Unit (up to a maximum of 2)
- Additional IBM 1816 Printer Keyboard (up to a maximum of 2)
- Additional IBM 1053 Printer Units (up to a total of eight 1053s and 1816s)

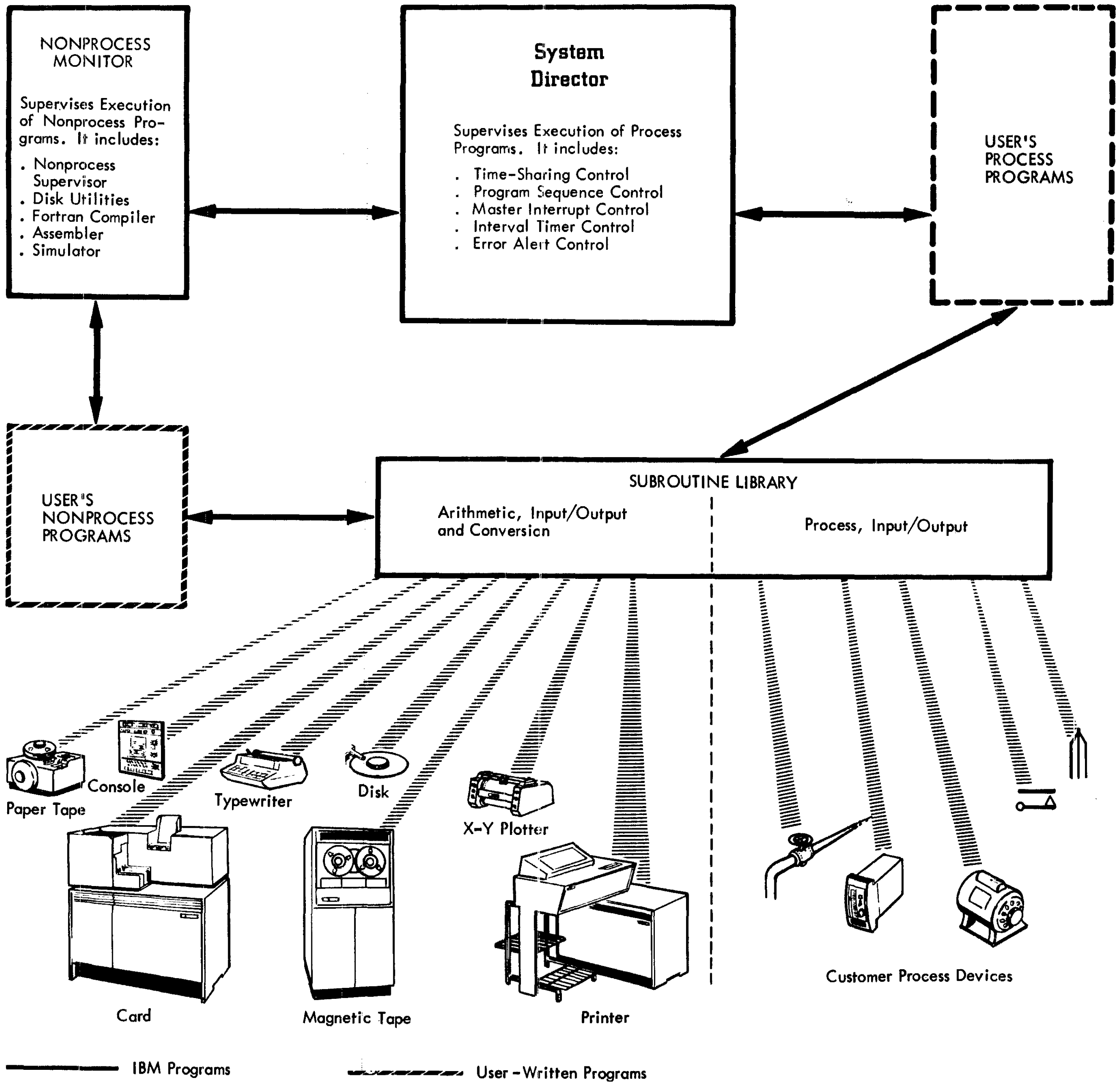


Figure 1. IBM 1800 Time-Sharing Executive System

- Additional Data Channels (up to a total of 9)
- Additional Interrupt Levels (up to a maximum of 24)
- Multiplexer Unit (Solid state and Relay)
- Analog-Digital Converter (up to a total of 2)
- Digital-Analog Output
- Digital Input
- Comparator
- IBM 1443 Printer Unit
- IBM 2401-2402 Magnetic Tape Units (maximum of 2)
- IBM 1627 Plotter Unit
- IBM 1054 Paper Tape Reader
- IBM 1055 Paper Tape Punch

MODES OF OPERATION

The IBM 1800 Time-Sharing Executive System consists essentially of two main parts: (1) a Skeleton Executive and (2) a Nonprocess Monitor. It is through the Skeleton Executive that process control and data acquisition applications are serviced in the on-line mode, while the Nonprocess Monitor operates either in the time-shared mode or as an independent programming system to provide data processing functions in a standard off-line mode. Each of these modes is brought into play by an appropriate and corresponding system generation procedure. The user elects the option of constructing an on-line or off-line system tailored to individual requirements.

On-Line Mode

In real-time processing, inputs arrive randomly from a process being monitored to the processor-controller. The computer rapidly responds to each input usually by conveying an output back to the process. This is in contrast with conventional batch processing where groups of input data are processed by passes through the computer. The notion of real-time usually implies that a processor-controller is responding to inputs as they occur in the physical world.

TSX operates in this mode under the control of the Skeleton Executive. In an on-line environment, user-written programs may monitor and/or control a process operation at any time. The machine is also permitted to be shared by process and nonprocess work: that is, batch work may be interleaved with other work. Whenever variable core is not required for a process program, the Nonprocess Monitor may be brought into service. All core loads and/or programs executed are accessed from the system resident disk cartridge.

Off-Line Mode

The off-line TSX system operates in this mode under the control of the Temporary Assembled Skeleton (TASK) as a dedicated Nonprocess Monitor System. Typical off-line operations are assemblies, compilations, disk utility functions, and the execution of data processing programs.

An off-line system can be used to test problem programs before they are permanently stored and catalogued on the system cartridge, to execute problem programs that require the full capacity of available disk drives for data files, or to execute problem programs that are used so infrequently that their on-line storage is not justified. It is also used to build an on-line disk resident system.

SYSTEM CONCEPT

ROLE OF THE SKELETON EXECUTIVE

The Skeleton Executive constitutes the framework of an on-line TSX system, and must be resident in permanent core storage before any continuous and coordinated real-time processing can take place. Other portions of the system are brought into core from disk storage as they are required to perform specific functions.

The Executive is extremely flexible and can be assembled at system generation time so that no core is wasted by selecting any of the numerous options available. The user may include frequently-called subroutines, fast response interrupt servicing routines, and other user-written programs in the skeleton to make the most effective use of his control system.

A typical skeleton executive might consist of the following parts as shown in Figure 2. The function of each individual component will now be described.

Skeleton I/O. This is a set of input-output subroutines which provides a rapid and easy method for the user to reference the various data processing input-output devices (e.g., card read punch, disk, printer) for input or output of data. It includes:

- DISKN (Disk Storage Subroutine - performs all reading from and writing to the IBM 2310 Disk Storage Unit)
- TYPEN/WRTYN (Printer-Keyboard Subroutine - transfers data to and from the IBM 1053 and IBM 1816 Printer-Keyboard)
- PRNTN (Printer Subroutine - handles all print and carriage control functions relative to the IBM 1443 Printer

These and other basic system routines make up the Skeleton I/O package which corresponds to an identical set of input-output subroutines used by TASK. A description of each subroutine will be found elsewhere in the TSX Systems Reference Library.

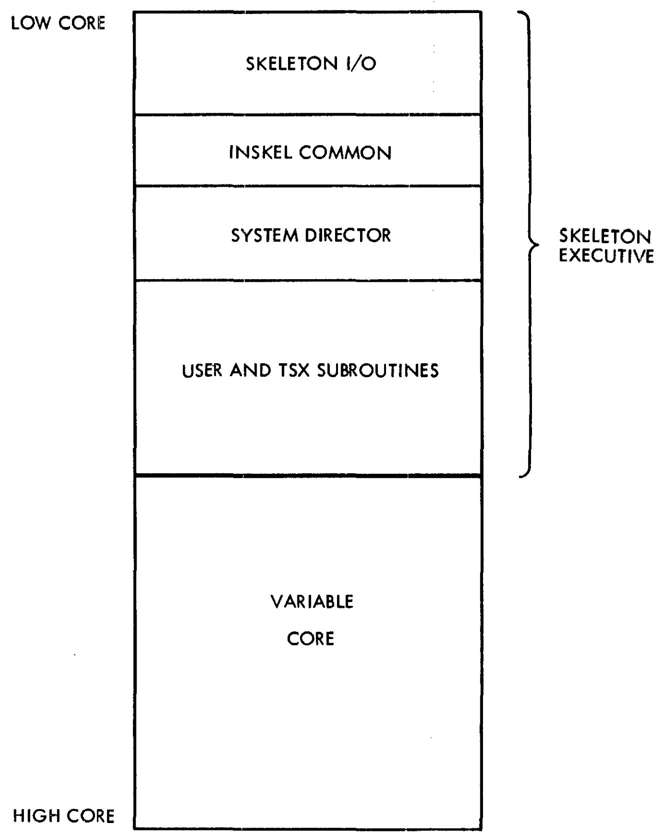


Figure 2. A TSX On-Line System -- Illustrating the Skeleton Executive

INSKEL COMMON. A uniquely labelled common area in the skeleton set aside for communications among the various types of core loads used in the system. It can be referenced by any process or nonprocess program under the on-line system.

System Director. This is the nucleus of the Skeleton Executive and controls all facets of process monitoring. It directs the handling of interrupts in a priority fashion determined by the user; supervises the execution of any number of mainline core loads or programs dictated by the process; services all error conditions with a minimum of disturbance to most processes under control; maintains the 1800 interval timers; and makes the system available to the Nonprocess Monitor.

User-Written Programs. The user has the option to include as many programs and subroutines as possible in the skeleton for reasons of frequent usage, rapid response, and optimum utilization of disk space. These may take the form of:

- Interrupt subroutines
- Timer subroutines
- Count subroutines
- Special trace and error subroutines
- IBM-supplied arithmetic, I/O, and other subroutines
- Any other user-written subroutines

These are first compiled/assembled in relocatable format and stored on disk; at skeleton build time, they are relocated into the Skeleton Executive.

TIME-SHARING

In many industrial installations, the real-time control system will not utilize all the computer time; therefore, time will be available to perform background jobs. Time-sharing techniques can thus be employed when idle processor-controller time is available in a given system environment to offer the user the kind of service he requires. The notion of time-sharing also implies the sharing of computer resources, since not only time but primary and secondary storage as well as most input-output facilities are also shared.

When idle time is available in the IBM 1800 TSX System, control can be automatically transferred to an independent Nonprocess Monitor System which is identical to any stack-job monitor system. All assembling, compiling, simulating, and other system activities can now be executed under the control of the Nonprocess Monitor. Performing such jobs time-shared has a distinct advantage in that any time not required for process control functions can be used for data processing functions. Also, since process control programs and strategies tend to change, time-sharing makes it extremely desirable to be able to modify these programs and strategies at the on-line installation without taking the computer off-line. It is through the time-sharing feature that the utilization of the 1800 system is best optimized.

VERSATILITY IN SYSTEM CONFIGURATION

A modern real-time operating system must be geared to change and diversity. The TSX system itself can exist in an almost unlimited variety of machine configurations: different installations will typically have different configurations as well as different applications. Moreover, the configuration at a given installation may frequently change. If we look at application and configuration of an operating system, we see that the operating system must cope with an unprecedented number of environments. All of this puts a premium on system modularity and flexibility.

TSX gives the user the ability to define his configuration according to his exact needs: he is therefore never bound to a fixed system. Furthermore, after having specified and generated a particular system, he is still free to move process and/or nonprocess portions of his system from one disk storage device to another.

In general, the input-output capability of the IBM 1800 Data Acquisition and Control System can be backed up. For example, under program control, a 1053 Printer can have its messages automatically switched to a back-up 1053 Printer; disk storage drives can be logically switched or removed from the system; and any device may be removed from service if it continues to fail. This dual capacity indicates that an installation may suffer from the failure of one or more input-output devices, and remain "on the air" under the most stringent usage conditions. Hand-in-hand with this back-up capability, a history of hardware device failures can be examined at any time for maintenance purposes.

CONCEPT OF A CORE LOAD

In practice, the core storage size of a data acquisition and control system is not sufficient to contain (nor does it need contain) all of the instructions required for the execution of all functions at any one time. Thus, the entire set of instructions must be broken down into smaller pieces, and these pieces be made available for immediate loading. To facilitate rapid loading, they should be stored on disk in executable core image format.

The technique of program segmentation is employed in the TSX system where programs are formed into smaller units called core loads. A core load is, by definition, an executable program or portion of a program which performs some user function. It is not necessarily a program in its entirety because this program may well be too large to fit into variable core in one piece for execution. The core load is unique in that it is stored on disk in core load core image format to facilitate rapid loading when it is called for execution.

Figure 3 illustrates the four types of core loads commonly used in TSX. A core load may contain other subroutines that are not associated with the main program - that is, subroutines not otherwise available in core (either included in the skeleton, or in the form of load-on-call subprograms). A typical core load may consist of a mainline or interrupt program, in-core interrupt subroutines, and all other required subroutines that are not included with the Skeleton Executive.

Core loads are important in real-time systems for the following reasons:

- Real-time linkages are automatically built
- The core-load is permanently built and stored on disk for rapid execution
- Core loads are called by name
- No compiling/assembling is needed at execution time.

LOCAL SUBPROGRAMS

TSX provides a facility for loading subroutines at the time they are called for in the execution of a program. Such a subroutine is known as a LOCAL (load-on-call). All LOCALs called by the same mainline program in a core load use the same area of core storage by overlaying one another as they are called. A copy of each LOCAL subprogram used with a core load is kept on disk in core-image format together with that core load (see Figure 3).

LOCALs thus allow the user to have, effectively, a larger program executed in core than would otherwise be possible if all the subroutines were loaded into core at the same time. There is no theoretical limit to the number of LOCALs which the user wishes to implement. This means a virtual extension of variable core. Other advantages of this feature are (a) the ability to read in subroutines, and (b) the breakdown of core loads to the subroutine level.

processing programs during their execution. A key control program is the System Director which is loaded into main storage (as part of the resident Skeleton Executive) and remains there indefinitely to ensure continuous coordinated operation of the system. Other parts of the system are brought into main storage from secondary storage as they are required to perform specific functions. Processing programs consist of language translators and service programs that are provided by IBM to assist the user, as well as problem programs that are user-written and incorporated as part of the TSX system. Both IBM and user programs have the same functional relationship to the control programs.

CONTROL PROGRAMS

There are three control programs within the TSX system:

Temporary Assembled Skeleton (TASK)
System Director
Nonprocess Director

Temporary Assembled Skeleton (TASK)

TASK is a stand-alone disk oriented monitor program from which an on-line or off-line TSX system is constructed. It performs three distinct functions:

- Supervises the generation of a disk oriented TSX operating system according to user specifications.
- Supports a full monitor capability so that TSX can be used as a data processing monitor system.
- Allows the user to load absolute programs into core for execution, or to store them on disk.

Since real-time process control installation requirements vary from installation to installation, it follows that each installation must be defined or tailored to the specific system functional requirements and input-output configuration of that installation. The tailoring function, defined as system generation, is carried out by TASK which provides the facilities for the creation and maintenance of a monitor system composed of IBM and user-written programs. The user specifies his system through the medium of equate cards.

Figure 4 illustrates TASK organization in simplified form.

The System Director

This control program forms the heart of the TSX system and resides in core storage at all times as part of the skeleton where all permanent areas are storage-protected to ensure that they are not inadvertently violated or altered.

The System Director directs the handling of process and data processing input-output interrupts, provides timer control over the process, is responsible for the orderly transfer of control from one core load to the next, and handles the transfer of control between the on-line and off-line modes. All process core loads are in core-image format on disk and are accessed at disk read speed.

The Director is read from disk only during a cold start or reload (EAC) operation. Primary entry to the System Director results from 1) internal and external hardware interrupts, 2) TSX calls from user's programs, and 3) errors.

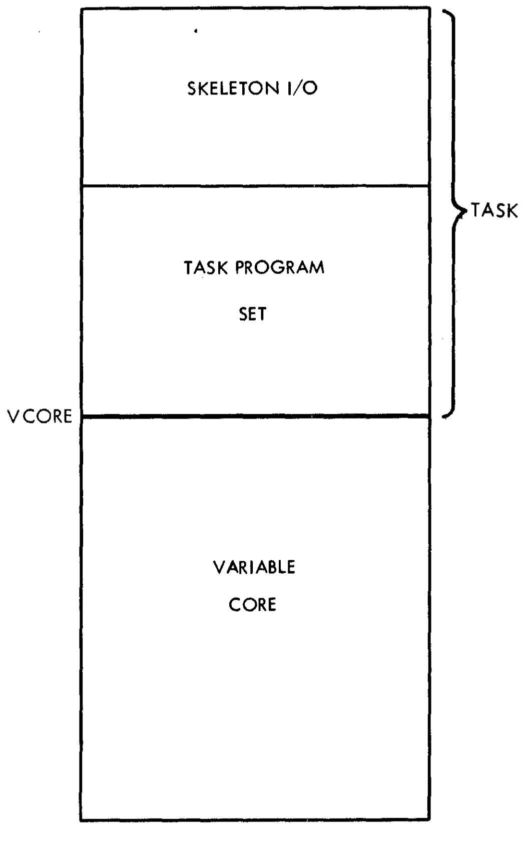


Figure 4. TASK Organization

INSKEL COMMON has already been defined. To assign a variable to this area, a special FORTRAN statement, COMMON/INSKEL/, must be used. All other attributes of the COMMON statement remain the same. This area must be used for communications between

- Core loads of a different type
- Interrupt core loads
- Combination core loads (if either is executed as an interrupt core load)
- A special core load and the mainline core load that calls it
- A mainline core load (which called a special core load) and the core load that restores it
- A skeleton subroutine and any other subroutine or core load

The normal COMMON area located at the high-address end of core storage can be referenced only by mainline or nonprocess core loads. The normal COMMON statement in a mainline, special, or nonprocess core load is used to refer to this area. This area is saved and restored when special core loads or time-sharing operations are initiated or terminated; i.e., communication between nonprocess core loads is possible.

The third area for COMMON is used only for interprogram communication for programs that form an interrupt core load or, between combination core loads when they are executed on the mainline level. The normal COMMON statement in an interrupt or combination core load is used to refer to this area. The highest addressed location of this area must be assigned by the user at system generation time, and must be an even number. This assigned location is the high-address boundary of the variable core storage area that is saved when an interrupt core load is loaded for execution. Thus, it is necessary to save only the area specified by the user for interrupt core loads (not the whole variable area).

MULTI-LEVEL PROGRAMMING

The interrupt structure of the 1800 system consists of a total of 24 hardware levels with up to 16 interrupt signals per level. These can, of course, be processed in a true priority sequence. A higher level interrupt subroutine will always interrupt a lower level interrupt subroutine, but beyond this, the Skeleton Executive permits interrupts to be "recorded" now for later processing.

The interrupt scheme within the Skeleton Executive also provides a great amount of flexibility and frees the user from most of the problems of servicing interrupts. Interrupt servicing subroutines may be assigned in the following ways:

1. An interrupt subroutine which must be executed immediately under any condition whatsoever can be made a permanent part of the skeleton. That is, the subroutine will always be in high-speed core storage and will be executable in the order of microseconds.
2. Those subroutines which are associated with a given mainline program can be assigned in such a way that they are always read into core storage with that mainline when it is loaded from disk. The response time of a mainline interrupt routine is almost the same as that of a skeleton interrupt routine only if the mainline core load containing the interrupt routine is in core when the interrupt occurs.
3. For low-priority subroutines, a core overlay technique allows the user to call an interrupt core load, bring it into core storage, save what was in core storage, and on completion of the interrupt process, restore core storage to its original state. These multiple operations of sequencing, saving, and replacing of core storage is automatically handled by the Skeleton Executive. All that is required of the user is to assign the priority. It should be mentioned that the priority interrupt sequence can be changed, at will, under program control.

The interrupt core load response time depends on the size of the core load and the disk layout. It is slower than the skeleton or mainline core load interrupts.

SYSTEM COMPONENTS

TSX components can be considered under two separate group-headings: (1) control programs and (2) processing programs.

In general, control programs govern the order in which processing programs are executed, and provide services that are required in common by the processing programs during their execution. A key control program is the System Director which is loaded into main storage (as part of the resident Skeleton Executive) and remains there indefinitely to ensure continuous coordinated operation of the system. Other parts of the system are brought into main storage from secondary storage as they are required to perform specific functions. Processing programs consist of language translators and service programs that are provided by IBM to assist the user, as well as problem programs that are user-written and incorporated as part of the TSX system. Both IBM and user programs have the same functional relationship to the control programs.

CONTROL PROGRAMS

There are three control programs within the TSX system:

Temporary Assembled Skeleton (TASK)
System Director
Nonprocess Director

Temporary Assembled Skeleton (TASK)

TASK is a stand-alone disk oriented monitor program from which an on-line or off-line TSX system is constructed. It performs three distinct functions:

- Supervises the generation of a disk oriented TSX operating system according to user specifications.
- Supports a full monitor capability so that TSX can be used as a data processing monitor system.
- Allows the user to load absolute programs into core for execution, or to store them on disk.

Since real-time process control installation requirements vary from installation to installation, it follows that each installation must be defined or tailored to the specific system functional requirements and input-output configuration of that installation. The tailoring function, defined as system generation, is carried out by TASK which provides the facilities for the creation and maintenance of a monitor system composed of IBM and user-written programs. The user specifies his system through the medium of equate cards.

Figure 4 illustrates TASK organization in simplified form.

The System Director

This control program forms the heart of the TSX system and resides in core storage at all times as part of the skeleton where all permanent areas are storage-protected to ensure that they are not inadvertently violated or altered.

The System Director directs the handling of process and data processing input-output interrupts, provides timer control over the process, is responsible for the orderly transfer of control from one core load to the next, and handles the transfer of control between the on-line and off-line modes. All process core loads are in core-image format on disk and are accessed at disk read speed.

The Director is read from disk only during a cold start or reload (EAC) operation. Primary entry to the System Director results from 1) internal and external hardware interrupts, 2) TSX calls from user's programs, and 3) errors.

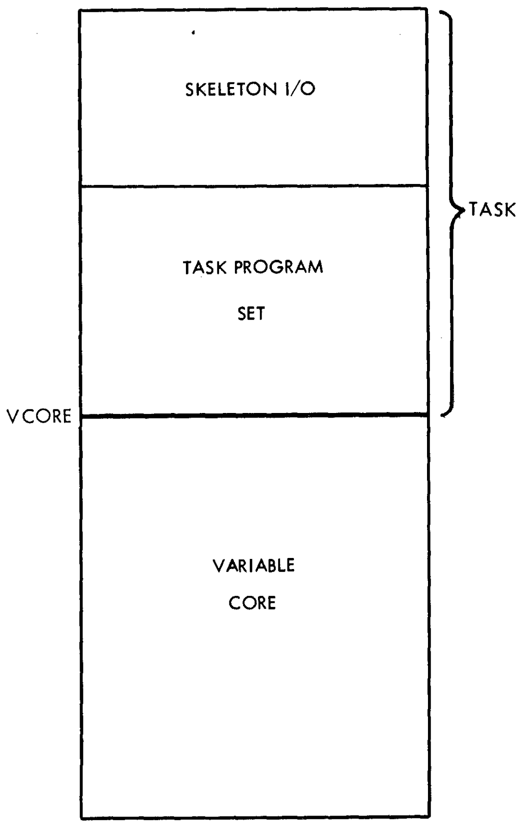


Figure 4. TASK Organization

The Nonprocess Supervisor

The Nonprocess Supervisor directs the execution of all nonprocess core loads which may be either IBM-supplied as part of the TSX package or user-written. It normally operates in the time-sharing mode under the control of the System Director, but it may also be run as a dedicated off-line monitor system under TASK.

Its main function is to recognize certain system control cards and transfer control to the processing program specified. It also initializes the nonprocess system whenever a job control card is identified.

PROCESSING PROGRAMS

Processing programs consist of service programs and language translators broken down as follows:

Service Programs

Cold Start Program
System Loader
Core Load Builder
Skeleton Builder
IBM TSX Subroutine Library
Disk Utility Program (DUP)

Language Translators

Assembler
FORTRAN Compiler
Simulator

Service Programs

Service programs include a group of loaders and builders which serve as system generation aids, as well as a disk utility program and a comprehensive IBM TSX Subroutine Library.

Cold Start Program

This program loads the Skeleton Executive into core, storage protects it, starts the real-time clock and calls the user's initial core load for execution. This operation places the System Director in control of the on-line system.

System Loader

The primary functions of the System Loader are to load the initial IBM TSX system onto the disk, build an interrupt assignment table from user-supplied control records, and prepare the disk layout for system operation. System assignment cards are used to inform the System Loader of interrupt level assignment of I/O devices, interval timers, and process interrupts. The loader makes entries in a directory called the Location Equivalence Table (LET) for each component part of the IBM TSX system.

Core Load Builder

The Core Load Builder program combines a user-written relocatable program together with all referenced subroutines not included in the Skeleton Executive into an executable core load for storage in the Core Load Area on disk. Core loads may be of several types: process mainline, combination, interrupt, or nonprocess.

All process core loads must be built and stored on disk prior to execution under control of an on-line TSX system. Input to the Core Load Builder is supplied by the user in the form of control records which contain the names of relocatable mainline programs, interrupts to be recorded, data files used, interrupt routines included as part of the core load, and LOCAL (load-on-call) subprograms.

Using the data provided by the System Loader and the Skeleton Builder, as well as information from programs and subroutines, the Core Load Builder establishes all subroutine linkages, hardware interrupt servicing linkages, and the creation of certain communications areas which are integrated with instructions to make up a core load.

Skeleton Builder

The Skeleton Builder program obtains its input from user-assigned control records, programs, subroutines, and information from the System Loader to construct the System Skeleton in core-image format which is then stored on disk. The skeleton is read into core for execution by a cold start operation. The rebuilding of the skeleton is required whenever routines are added or deleted, or other modifications are made. It is the System Skeleton which constitutes the Skeleton Executive.

IBM TSX Subroutine Library

This comprises a comprehensive set of reentrant subroutines as well as a select set of non-reentrant subroutines designed to aid the user in making efficient use of the IBM 1800 Data Acquisition and

Control System. The library contains the following:

- Data processing and process input-output subroutines
- Conversion subroutines
- Arithmetic and functional subroutines
- FORTRAN input-output subroutines
- Miscellaneous subroutines

Data Processing and Process I/O Subroutines. Data processing (printers, punches, etc.) and process input-output (P I/O) subroutines provide a quick and straightforward method for the programmer to reference the various data processing, digital and analog I/O devices for input or output of data. All I/O routines may be called directly from FORTRAN: data processing I/O subroutines may be called indirectly by the use of FORTRAN I/O.

Conversion Subroutines. The design and operation of the numerous input-output devices is such that many of them impose a unique correspondence between character representations in the external medium and the associated bit configurations within the computer. Conversion subroutines convert inputs from these devices into a form in which the computer can operate and to prepare computed results for output on various devices.

Arithmetic and Functional Subroutines. The arithmetic and functional group of subroutines includes a selection of twenty-seven basic routines which are most frequently required because of their general applicability. The arithmetic library contains both the routines that are visible to the FORTRAN programmer, as well as the many routines that are used by the FORTRAN compiler generated object code, which may also be used by the Assembler programmer.

FORTRAN I/O Subroutines. FORTRAN I/O subroutines provide a link between the FORTRAN object code and the I/O devices. They support both standard and extended precision.

Miscellaneous Subroutines. The miscellaneous group provides the user with the ability to perform certain machine operations using the FORTRAN language. These include real-time, selective dump, trace, and overlay routines.

Real-time subroutines are operational control routines which service the Skeleton Executive in an on-line environment. Examples are TIMER (specify one of two hardware interval timers for some periodic activity), LEVEL (set one of twenty-four levels for programmed interrupt use), and MASK (inhibit selectively one or more levels of interrupt).

Selective dump subroutines allow the user to print chosen areas of core storage during the execution of an object program. For example, DUMP will output on the list printer, in hexadecimal or decimal format, a certain portion of core storage; DUMPS will print the status of the 1800 (that is, status indicators, contents of registers, and work areas).

The user may exercise the option of writing his own mainline trace interrupt routine which can be included in a core load to process a trace interrupt. He might, for example, design such a routine to monitor a number of conditions. The subroutine TRPRT is available for use in tracing routines which print a specified number of characters on the 1053/1816 Keyboard Printer or 1443 Printer.

The TSX Subroutine Library also contains an overlay routine called FLIP which serves to call LOCAL (load-on-call) subprograms into core storage. All LOCALs in a given core load are executed from the same core storage locations; each LOCAL group overlays the previous group.

In order to permit entry from multiple programs and interrupt levels before completing computations from a previous call, the arithmetic and functional subroutines, and most of the I/O subroutines, are designed to be reentrant. That is, they can be entered from a different level of machine operation despite the fact that they may not have completed operation on a previous level. Non-reentrant versions of the arithmetic, functional, and conversion subroutines are also supplied.

Disk Utility Program (DUP)

The Disk Utility Program is a comprehensive group of generalized utility and maintenance routines designed to aid the user in the day-to-day operation of the TSX system. By this means, the most frequently required services of disk and data maintenance can be performed with a minimum of effort. The TSX DUP philosophy is to provide as much assistance as possible to the user. DUP is a component part of the Nonprocess Monitor.

DUP is called into service by the Nonprocess Monitor Supervisor (SUP) whenever it recognizes a DUP monitor control card. It is also automatically summoned after the successful completion of an assembly or FORTRAN compilation. DUP functions can be summarized as follows:

1. It permits the user to store, modify, and refer to programs and data using the compact and economical direct-access disk storage facilities of the system without regard to specific input-output configurations.
2. It allows the free interchange and use of programs and data among programmers.
3. It provides a systematic method to identify and locate programs and data, and systematic methods to reference data after it is located.

All of these functions can be carried out while the TSX system is on-line, as well as in the off-line mode. Examples of DUP facilities include the following:

- Change sequence of execution of core loads
- Replace a core load with another core load
- Create disk files
- Replace an object program already stored on disk
- Build core loads (in conjunction with the Core Load Builder)
- Define the disk configuration
- Dump data/program from one medium to another
- Delete a program, core load, or a data file from the disk
- Pack a file on the disk to eliminate unused areas, thus minimizing disk storage requirements
- Modify core loads on-line

Language Translators

Language translators assist a programmer by enabling him to define a problem or an application in a language form that can be readily learned and understood. In the TSX system, the user may define his problem solution or application

In a flexible easy-to-use symbolic language – Assembler language, and/or
In a form of mathematical notation – FORTRAN

Assembler

The Assembler program is a one-for-one disk oriented symbolic type translator which assembles object programs in machine language from source programs written in symbolic language. It normally resides on disk. The assembler accepts control records and source programs in card form only. Upon a successful assembly, the object program in relocatable format is moved to disk where it is permanently stored, or, alternatively, called for execution. The Assembler Language is fully described in the publication IBM 1800 Assembler Language, Form C26-5882.

FORTRAN Compiler

The FORTRAN Compiler translates programs written in the FORTRAN language into executable machine language. The real-time TSX FORTRAN Compiler permits the user to make the most of the digital and analog I/O features using a higher level language, while at the same time allowing background jobs to be executed. Since FORTRAN is easily understood by technical personnel, its availability in the TSX system reduces significantly the programming effort required. For a full description of the FORTRAN language, see IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715.

Simulator

The Simulator is designed as a debugging aid which allows the user to check out or test process and/or nonprocess programs without disrupting normal TSX system operation - that is, without taking the system off line. It functions under the control of the Nonprocess Monitor.

FUNCTIONS OF EXECUTIVE PROGRAMS

This section describes the functions of the three main executive programs which constitute an IBM 1800 Time-Sharing Executive System, namely,

Temporary Assembled Skeleton (TASK)
System Director
Nonprocess Monitor

and discusses the concepts underlying their use. Sample programs and coding are interspersed throughout the text as demonstration of good programming practice and technique. Since the Temporary Assembled Skeleton (TASK) is the first program with which the user becomes involved in the creation of an on-line or off-line TSX system, it is discussed at the outset.

TEMPORARY ASSEMBLED SKELETON (TASK)

It has already been mentioned that TASK (Temporary Assembled Skeleton) is a builder or "tailor" card monitor system with strong disk capabilities from which an off-line or on-line TSX system is constructed. The use of TASK, therefore, constitutes the intermediate stage in system generation towards placing a system on-line. In an on-line TSX system, TASK control ceases at cold start time when the System Skeleton has been loaded into core storage. In an off-line TSX system, TASK itself functions in much the same fashion as a System Skeleton with permanent time-sharing.

For simplicity, TASK can be considered in two parts (see Figure 4):

- Skeleton I/O
- TASK Program Set

Skeleton I/O

The Skeleton I/O is a collection of input-output and general supporting subroutines that the TSX system requires to be in core at all times. It is that portion of a user-configurated TASK which corresponds exactly to the Skeleton I/O on an on-line TSX system.

Figure 5 illustrates this correspondence, as well as the core layout, at two time periods of an on-line and an off-line system.

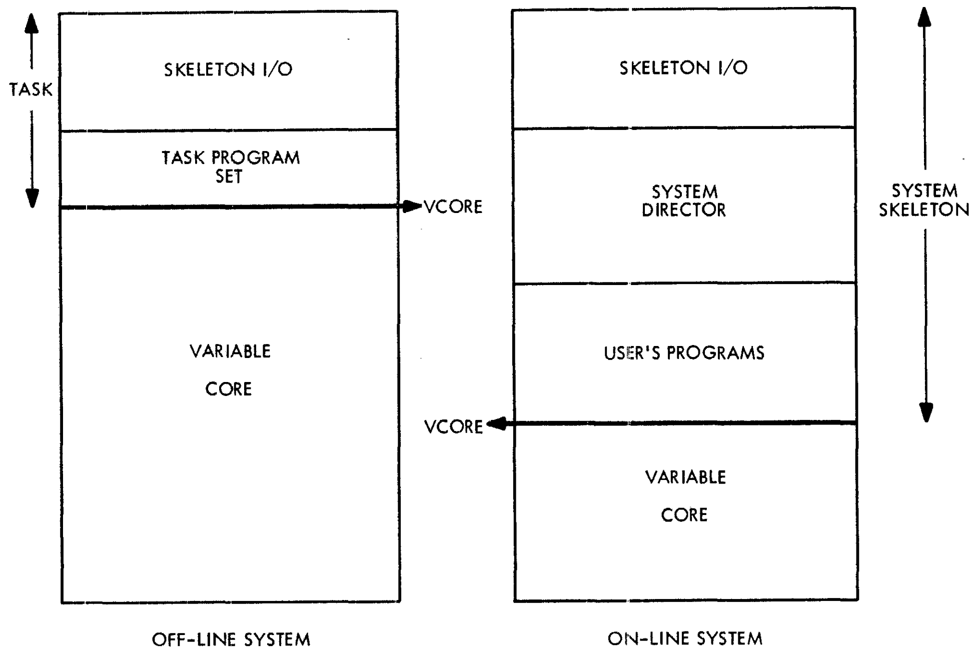


Figure 5. Correspondence between TASK and the System Skeleton

The I/O routines used by TASK form the basis of the Skeleton I/O. These consist of the following:

- DISKN — Disk subroutine as used by TSX Operating System
- TYPEN/WRTYN — Printer/Keyboard subroutine as used by TSX Operating System
- PRNTN — Printer subroutine as used by TSX Operating System
- CARDN — TASK only Card I/O subroutine

A description of each of the above subroutines will be found in the TSX Systems Reference library.

Since the TSX system requires that at least one disk be present on the 1800, DISKN must be in core at all times. If the user has assigned a 1053 or 1816 to his machine, TYPEN/WRTYN must also reside permanently in core. Although CARDN is in core during TASK execution, it does not normally form a part of the Skeleton I/O. The user must therefore define whether or not CARDN is to be a component part of his skeleton by means of the TASK equate card CDINS. If it is not, CARDN automatically becomes a part of the TASK Program set. It is through the Skeleton I/O that an off-line system obtains full monitor capabilities.

TASK Program Set

The TASK Program Set is that integral part of the Temporary Assembled Skeleton which functions in a similar manner to the System Director. It consists of:

- TASK Master Interrupt Control (TMIC)
- TASK Director
- TASK Error Alert Control (TEAC)
- Absolute Loader function
- Load Monitor function
- Skeleton Build function
- TASK Conversion routines
- TASK Utilities

TMIC directs all I/O interrupts to their corresponding servicing routines and resets all process interrupts, while TEAC processes errors that have been found by other parts of TASK. The TASK Director initializes TASK and directs the execution of the Absolute Loader function, Load Monitor function, and the Skeleton Build function.

The Absolute Loader gives the user a facility to load absolute assembled programs from cards to core for execution. It can also be used for the storing of user-written programs or data on disk. The use of this function is discussed later in some detail (see Programming Techniques). The Load Monitor function serves to initialize the TSX Nonprocess Monitor for execution. There are two conversion routines: (1) TASK HOLEB converts hollerith input to one or two EBCDIC characters per word output, while (2) TASK EBPRT converts two characters per word EBCDIC input to two characters per word, system, list, or EAC printer code.

A complete utility package comprised of full trace, check/stop trace, four utility programs, and a utility monitor can be included within TASK at assembly time. The user elects this option through equate cards.

Except in the case of a skeleton builder option, a TASK disk load, or a cold start, TASK is loaded with a 4-card TASK high core loader.

For a more complete description of TASK functions and system generation procedures, the user is referred to the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Other considerations affecting the use of TASK are discussed under System Design Considerations.

THE SYSTEM DIRECTOR

The System Director is the nucleus of the skeleton executive of a TSX system, and always resides in core as part of the skeleton to direct the handling of interrupts, to load and execute core loads, to expand usage of interval timers, and to process errors. Primary entry to the System Director derives from internal and external hardware interrupts, TSX calls from user's programs and errors. Its principal component parts comprise the following:

Master Interrupt Control (MIC). This is a reentrant control program which automatically directs all internal, I/O, external, and programmed interrupts to their proper interrupt servicing routines. Control returns to MIC as long as unserviced interrupts exist.

Program Sequence Control (PSC). The Program Sequence Control Program is responsible for orderly transfer of control from one user-specified core load to the next. A core load may also temporarily be saved on disk pending the processing of another core load. All PSC functions are restricted to process mainline core loads.

Time-Sharing Control (TSC). This controls the time-sharing of variable core between process and nonprocess core loads by a core exchange method. TSC is entered selectively from the execution of a CALL SHARE statement or automatically by a CALL VIAQ statement when the queue is empty.

Interval Timer Control (ITC). ITC services all interrupts involving three machine timers A, B, and C, nine programmed timers, and a programmed real time clock. The programmed timers and the real time clock are based on timer C. Timer C is reset by the subtraction of a fixed value; accurate timing is therefore kept, even when the response to the timer interrupt itself may be delayed. It also services the "no-response routine" for the 1053/1816 printers in the Skeleton I/O. As an option, it also services the Operations Monitor during nonprocess execution. Periodic interrupts are generated from interval timers rather than from the real time clock. The programmed timers interrogate the Interrupt Core Load Table (ICLT), but only skeleton count routines are entered into. If there is no such routine, the condition is recorded for later servicing.

Error Alert Control (EAC). The EAC program resides in core at all times, and is called to process all error conditions whenever an error develops. EAC

- optionally saves core for future reference
- optionally branches to a user-written error subroutine (which may be included with each process core load) for further error analysis
- prints an error diagnostic message, and
- executes one of four possible error recovery procedures

Mainline Core Load Queue Table. This is a stack or pushdown list of names of mainline core loads (and their respective priorities) that have been queued (that is, put in line) for future execution.

Although the Queue Table forms part of the System Director, the real-time TSX queue-calling statements (e.g., QUEUE, UNQ, QIFON and VIAQ) are designed as subroutines which may be included in the skeleton or with the mainline at the user's discretion. Processing of a mainline is not suspended as a result of queueing a higher priority mainline.

Level Work Areas. A level work area of 104 words (in the skeleton) is required for

- each interrupt level used
- process mainlines
- nonprocess core loads, and
- internal errors

A level work area contains interrupt level instructions, MIC linkages, and work areas. It is used to allow recursive entry to those programs supplied by IBM.

Each of the following System Director functions will now be explained in some detail:

- Program Scheduling
- Handling of Interrupts
- Use of Interval Timers
- Use of Time-Sharing
- Use of the Operations Monitor
- Error Alert Control

PROGRAM SCHEDULING

Control processes may be classified under three basic headings:

Program or event sequence
Time dependence
Interrupt initiation

In practice, a process may be a combination of all three categories, but is usually weighted more heavily towards one. Rarely does a process lend itself to only one.

Figure 6 is a simplified version of a process based totally on program sequence. An example might be a crude-oil distillation unit in an oil refinery. A scan is made to see what the present status is, tests and calculations are made to verify the information, optimization towards a given product mix is applied, required changes to process variables are effected, data is recorded, etc. Each event thus depends on the completion of previous events.

A process based on time is illustrated by Figure 7. This classification could be applied to a process involving a solitary engine test stand. For example, a given throttle position and resistance load are set up. At specified time increments, one or more variables are recorded, such as manifold pressure, RPM, fuel flow, fuel level, oil temperature, oil pressure, etc. When all the variables have been recorded, the throttle position and/or load resistance are changed and a subsequent timing cycle initiated. Finally, when all specified combinations of throttle and load resistance settings have been tested, the system is initialized for another engine. Each event in this situation depends on time.

Note that in practice, the servicing of a process as depicted in Figure 7 is not necessarily sequential in nature. Also, it is the actual time period that schedules the servicing of an event. The manner in which servicing takes place is not dictated by the type of program (e.g., mainline, interrupt routine) which initiated the event.

The third classification is illustrated by Figure 8. An example might be the input phase of a hospital information system. With no input information, the system switches over to the time-sharing mode or remains idle. When, however, a patient enters the hospital, certain historical information is punched into cards. An interrupt is then initiated by an operator. The interrupt recognition routine sets up the card read program and the patient data enters the system -- the system then returns to time-sharing or to an idle condition. When, later, a doctor requests medication for a certain patient, in a specified quantity, at certain time increments and duration, he sets up the proper information on a manual entry unit and initiates an interrupt. The interrupt recognition routine again calls the appropriate program which reads in the manual entry, verifies the information, enters it in the specified files, and once again returns the system to the time-sharing mode. In a similar fashion, other input information such as records and/or schedules for dietary, patient status, laboratory, surgery, etc., are entered. Events thus classified are initiated by interrupts.

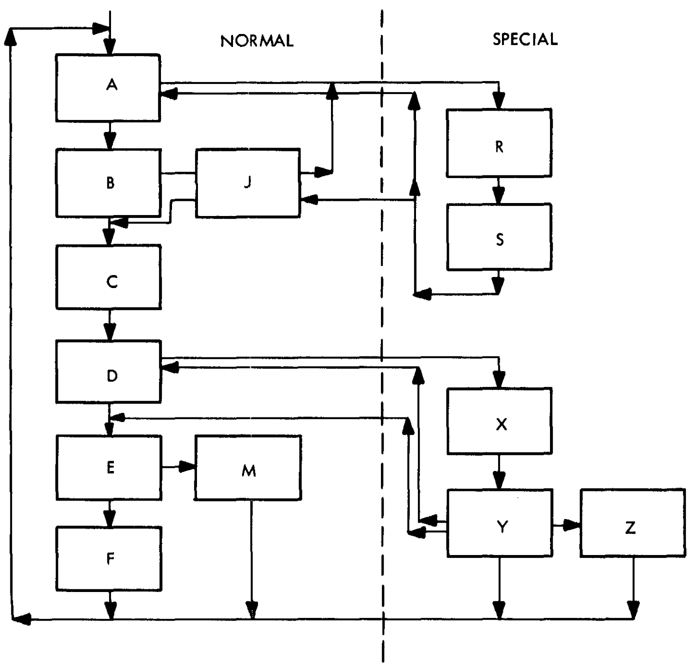


Figure 6. Program or Event Sequence

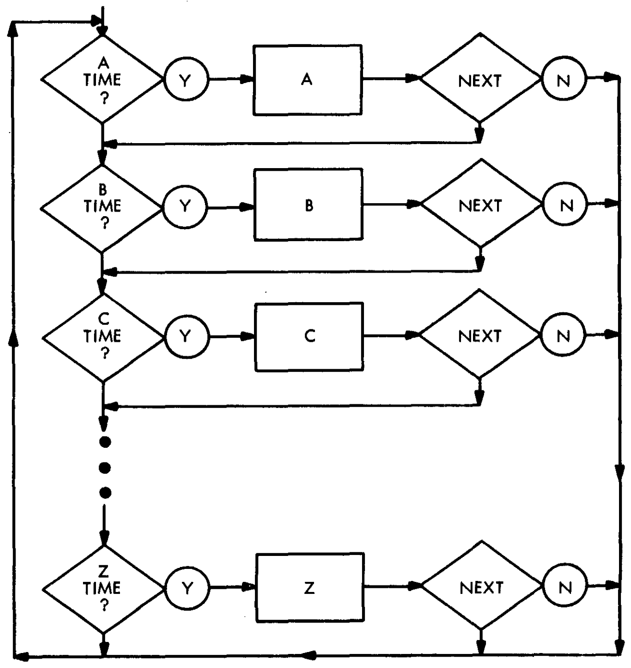


Figure 7. Illustrating Time Dependency

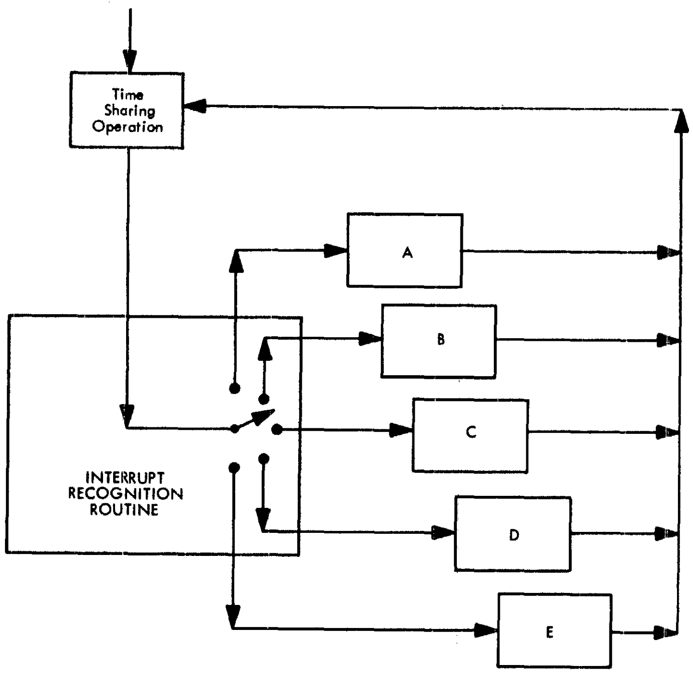


Figure 8. Interrupt Initiation

It is obvious from the foregoing that if each application illustrated were expanded to its complete operating requirements, it would most likely consist of all three classifications to some degree. For example, in Figure 6, an inventory log of input and output material is given every hour. This is required to update inventories, product costs, etc. Also, an interrupt will occur whenever a heating unit goes out of range. This will immediately initiate a program to rectify the situation.

Multi-Level Control. A control system must be able to immediately recognize certain situations of a physical process. It must also be able to ignore certain functions until they occur. In practice, the first requirement is more critical. In either case, the normal sequence of events will be interrupted until some action is taken. The situation is further complicated if a second interrupt, more critical than the first, occurs during the action phase of the first interrupt. The servicing of the first interrupt must obviously be suspended while attention is given to the more critical interrupt. Such a chain of interrupts may continue through several iterations as shown in Figure 9. Upon completion of the required action of each interrupt, the previously interrupted action must be continued until the system returns to normal. From this brief picture of multi-level operations, we see that program scheduling now becomes more complex. The user must now have the capability to take immediate action, record the occurrence for later action, or arrange for action to be taken as soon as possible, but on a less critical level. To do this, the user defines what is to be recognized on each level and sets this up by machine configurations. Later his program sets up when action is to be taken and at what level.

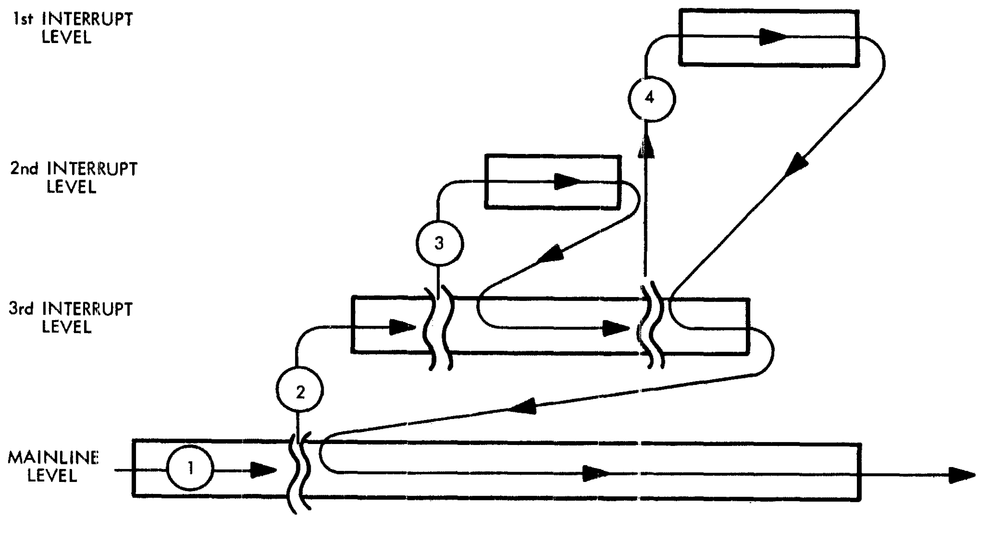


Figure 9. Multi-Level Processing

Program Scheduling Requirements

In a control system application, the scheduling of programs to be executed on the normal or mainline level constitutes the main problem. During certain phases of a control system, the user will execute programs in a set sequence. This type of sequence may be set up by a program condition, an interrupt, or a given time period. Sequencing or chaining of programs may or may not be required depending on the user's specific application.

A direct sequence or chain of programs is required for two separate situations. The first situation is a set of programs whose functions must be in a given order that cannot be interrupted except for critical conditions. The second derives from a program that is too large for core size available, so it must be segmented into several separate programs. These programs will now overlay each other, and must therefore be scheduled in a fixed sequence.

As illustrated in Figure 6, special sequences of programs may also be required on the mainline level under certain special conditions. These special sequences are required under three conditions which come under normal operation. The first requirement is a sequence or chain of events that is common to several different phases of a system. This is logically equivalent to a subroutine which is called by several programs: the main difference is that a chain of programs is now being scheduled instead of a subroutine. The next requirement occurs when a situation is bordering a critical point, but is still within the limits defined by the user. In this event, the user may want a warning, but has no real need to be alarmed. The third situation is similar except that the user is alarmed and cannot therefore proceed with the present sequence of programs until certain conditions are met. This is a common situation in process control where process inputs are not acceptable and a special scan is set up until valid variables are obtained. As a result, the normal calculation, optimization, etc., are delayed but will be resumed as soon as possible.

The requirements stated thus far are categorized under program sequence since they have a definite relationship and order. Three commands are used to implement sequence control:

1.	CALL CHAIN	(specify the next program to be executed)
2.	CALL SPECL	(terminate the program, save it on disk, and execute the next program).
3.	CALL BACK	(return control to a program which was partially executed).

Multi-process control, however, presents a new scheduling problem. Since one control system is used to control two or more processes, the definite relationship and order of programs is normally applicable within each process but not between processes. However, each process must be able to schedule its own programs in such a manner that the control system can handle all schedules. Also, because each process will normally contain its own unique program sequences, one type of scheduling problem does not necessarily eliminate another. It should also be understood that multilevel processing does not always dictate unrelated program scheduling: all possible combinations must be considered by a given program scheduling situation. The queueing technique itself will not produce such a system, but when combined with the priority technique, the system becomes flexible enough for any control system's requirements. Four commands permit this form of control:

1.	CALL QUEUE	(enter a core load into a waiting queue)
2.	CALL UNQ	(remove a core load from a waiting queue)
3.	CALL VIAQ	(call the highest priority core load waiting in the queue to be executed next in sequence).
4.	CALL QIFON	(interrogate recorded interrupts)

Program Sequence Control (PSC)

The center of the scheduling system is the Program Sequence Control (PSC) Program which is permanently resident in core in an on-line TSX system working under control of the System Skeleton. PSC is a means by which mainline core loads are loaded to core, and control transferred from one core load to another, according to user specifications. The user sets up his requirements when he uses a chain or sequence type CALL or a queueing-type CALL statement. PSC performs the following functions:

- Loads all mainline core loads
- Saves and reloads the special core load
- Initializes the ICL Table for each core load
- Tests for errors in calls to load programs

Chaining or Sequence Technique

Chaining or sequence-type call statements permit the programmer to control the order in which tasks are performed, interrupts serviced, and off-line jobs allowed. This control is important since the various levels of control are necessarily carried out in sequence and the order is critical. For example, an optimizing routine too large for core storage can be executed in segmented parts if the programmer has control over their sequence. Three call sequences are used in chaining: 1) CALL CHAIN, 2) CALL SPECL, and 3) CALL BACK. Note that core load names referenced by the CALL statement must also be specified in a FORTRAN EXTERNAL statement. A core load name cannot be the name of a component subprogram of that core load. Figure 10 illustrates the use of these call sequences.

Such statements may be freely embedded within process programs written in FORTRAN or in the Assembler language. Through the use of these commands, within programs, the programmer can control the frequency and order in which the various levels of control are performed. Even when the various levels are not performed on a regular basis, these commands allow control over the sequence. Of equal importance is the ease by which sequence is changed as the process control problem changes with time.

CALL CHAIN --Normal Call

When a given core load is called for execution, the user sets up the following statement:

CALL CHAIN (NAME)

where

CHAIN	=	Entry to PSC
and NAME	=	Name assigned by user to the next sequential core load to be executed

This normal call transfers control to PSC, thereby terminating the current mainline core load at its last logical statement. PSC then sets up a disk function to read in the next mainline core load specified by NAME into variable core, overlaying the present core load that contained the CALL CHAIN statement. The new core load thus destroys the previous core load. Once the core load is in core, the disk I/O routine reverts to PSC, which in turn passes control to the new core load.

CALL SPECL -- Special Call

The second type of core load sequence is similar to the CALL CHAIN, except that the current core load and its associated parameters must be saved. This is set up as follows:

CALL SPECL (NAME)

where

SPECL	=	Special entry to PSC
and NAME	=	Name assigned by user to a special core load to be executed next

The special call suspends execution of the current mainline core load and transfers control to PSC which saves the present variable core area and all required parameters, such as index registers, accumulator, extension, return address, and status. This information is written to the Special Save Area on disk. Once the save operation is complete, the disk I/O routine returns control to PSC. The operation proceeds from this point exactly as in a CALL CHAIN.

Note that only one mainline core load can be saved. Thus, if a CALL SPECL is used in a core

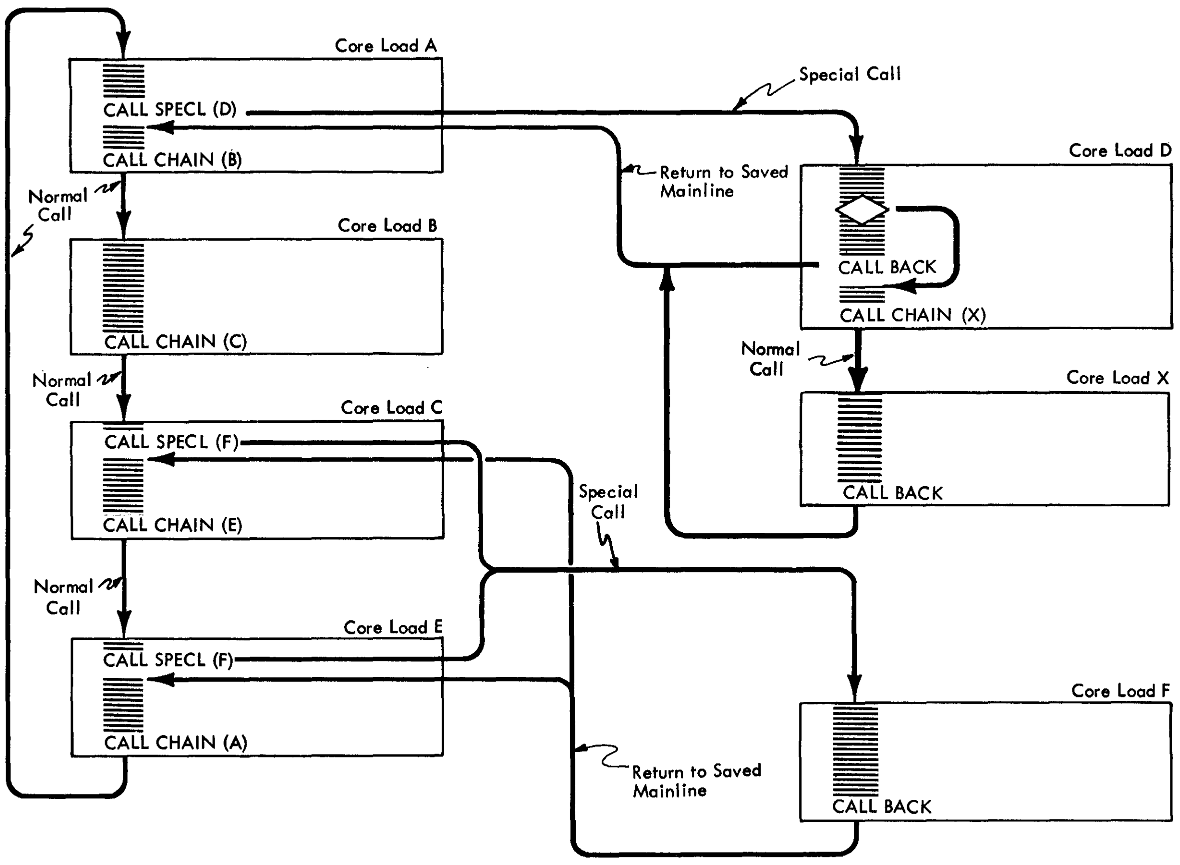


Figure 10. Use of Chaining (or Sequence-type) Call Statements

load that was referenced by another CALL SPECL, the mainline core load saved originally is lost. A core load called by a CALL SPECL may, however, chain to other core loads as long as these core loads do not contain a CALL SPECL (see Figure 10).

CALL BACK -- Return Saved Mainline

In order to return to the saved core load, a third call statement becomes necessary. This is used only in conjunction with the special sequence function. It is set up as follows:

CALL BACK

where

BACK	= Special entry to PSC

CALL BACK transfers control to PSC which, in turn, initiates a disk read operation to load variable core with the information stored in the Special Save Area on disk as the result of a CALL SPECL. When the read operation is complete, the disk I/O routine returns control to PSC. All saved parameters are now restored, and the restored core load continues execution at the saved return address (that is, the statement following the CALL SPECL statement).

It should be noted that a CALL BACK statement is required only if the saved core load is to be restored and continued. The user may well initiate a new core load as the result of a special core load. This new core load could then be referenced by a CALL CHAIN or a CALL SPECL.

A core load is terminated or suspended as the result of any of the three calls: CALL CHAIN, CALL SPECL, or CALL BACK. CALL CHAIN and

CALL BACK are the last logical statements executed in a core load. However, a CALL SPECL will not be last logical statement of a core load if a CALL BACK has been executed to restore the saved core load, and to continue execution following the CALL SPECL statement.

Queueing and Priority Techniques

Queueing techniques normally use statistical methods to control the number of queues. The rule that governs the input and output order in which waiting requests are serviced is usually based on an ordered-queue discipline -- that is, first-come, first-served. Since we are considering the use of only one queue, a first-come, first-served control is only valid for a given priority. Therefore, as several priorities are, in practice, required by most control system applications, a priority technique must be enforced. A priority level is one of the most common ways of classifying interrupt requests according to their urgency. Note, however, that the urgency may change as a function of the condition of the servicing system. For example, a request may be given a higher level as waiting-time increases. Priorities are assigned by the user to programs, processes, and functions. The queueing and priority control techniques employed combine to provide a flexible method completely acceptable for scheduling unrelated core loads. Although the call sequences to be described are referred to as queueing calls, both queueing and priority control are implied.

CALL QUEUE -- Insert into Queue

The first of four calls is used to place a core load entry in the Core Load Queue Table (see System Design Considerations: System Director), and to continue with the execution of the present function. The format of the call is:

CALL QUEUE (NAME, P, E)

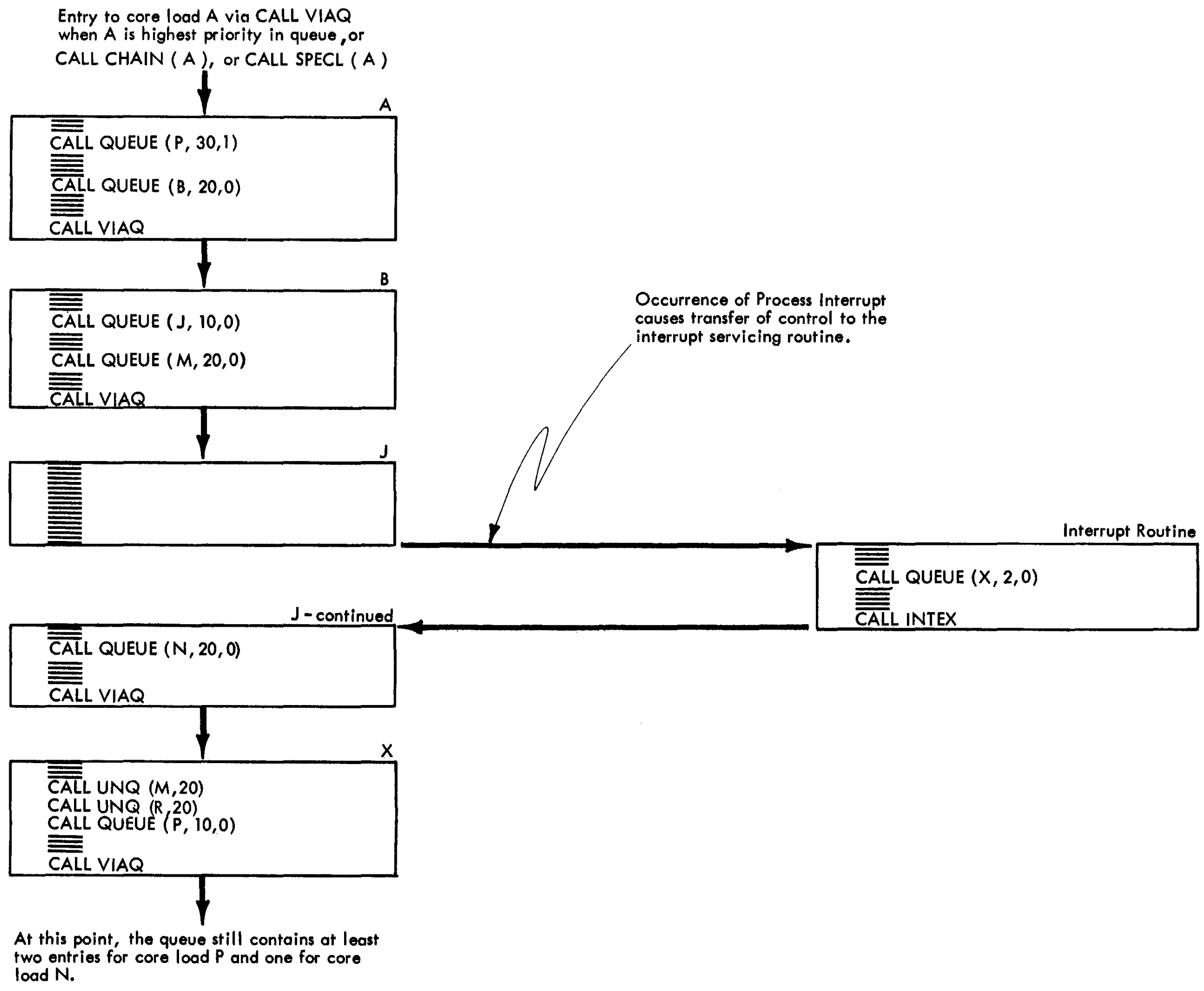
where

QUEUE	=	Name of the subroutine that places the specified core load in the Queue Table.
NAME	=	Name of user-assigned core load entry to be entered in the Queue Table (and in FLET).
P	=	Integer expression, specifying queue priority of core load NAME. This may be 1-32767. One (1) is the highest priority number.
E	=	Designated error procedure to be taken if the queue is full. In each case, an appropriate error message is printed (see Table 2: On-line EAC Errors and Recovery Procedures). The parameter is user-assigned as follows:

E	=	0. Ignore this call, and continue execution as if the core load had been queued.
E	=	1 through 32766. Replace the lowest priority entry currently in the queue with the name and priority specified in this call, if the priority of that current queue entry is lower (that is, numerically larger) than E. If there is no queue entry with a lower priority, execute the restart core load specified for this core load.
E	=	32767. Execute a restart core load.

In practice, E is always set to zero. The size of the Queue Table should be redefined by the user if it becomes saturated. The options listed under E (above) are provided by the Error Alert Control (EAC) program (described later).

Figures 11 and 12 illustrate the use of these functions. In Figure 12, an example is given of a series of mainlines which, if executed serially without interruption, would not allow queue testing for an inordinate amount of time. In order to be able to check the queue in some user-specified time period to see if any high priority core loads need to be executed, a program of the priority of the current executing program is queued; a CALL SPECL is then made to a core load that exits via a CALL VIAQ. The VIAQ routine then checks the queue for the highest priority program and executes it. When the executed program is the core load queued by core load A, a CALL BACK is performed which restores the original calling core load to execution status. This technique is commonly employed to break up the execution of a long program.



Entry to core load A via CALL VIAQ when A is highest priority in queue, or CALL CHAIN (A), or CALL SPECL (A)
A
CALL QUEUE (P, 30,1)
CALL QUEUE (B, 20,0)
CALL VIAQ
B
CALL QUEUE (J, 10,0)
CALL QUEUE (M, 20,0)
CALL VIAQ
J
Occurrence of Process Interrupt causes transfer of control to the interrupt servicing routine.
Interrupt Routine
CALL QUEUE (X, 2,0)
CALL INTEX
J - continued
CALL QUEUE (N, 20, 0)
CALL VIAQ
X
CALL UNQ (M,20)
CALL UNQ (R,20)
CALL QUEUE (P, 10,0)
CALL VIAQ
At this point, the queue still contains at least two entries for core load P and one for core load N.

Figure 11. Use of Queueing Statements

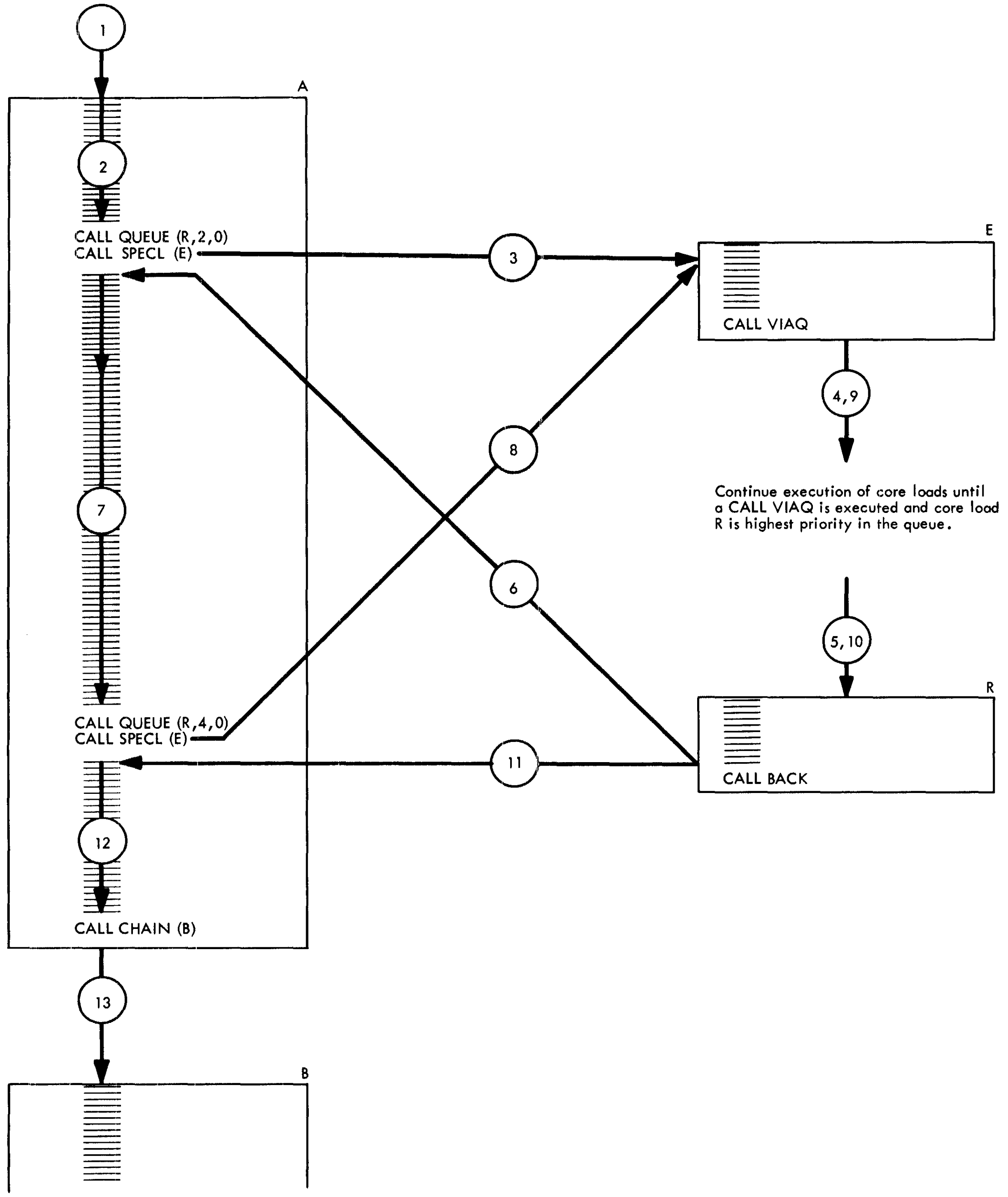
When a CALL QUEUE statement is executed, control is transferred to the real-time QUEUE routine which tests for an entry in the Queue Table with the identical name and priority as that specified in the user calling statement. If such an entry exists, a further entry will not be made -- a given core load and priority cannot, by definition, appear more than once in the Queue Table. However, the same core load with varying priorities may appear once for each unique priority.

If the entry is already in the queue, control is passed to the next executable instruction following the CALL QUEUE statement. If this is not the case, the QUEUE routine tests for a Queue-Table-full condition. If the table is full, control passes to EAC which executes the function specified by the E parameter. If the Queue-Table-full condition test is not satisfied, the QUEUE routine will place the core load entry in the Queue Table, and transfer control to the next instruction following the CALL QUEUE statement.

CALL QUEUE may be executed in a program that was initiated by an interrupt or a specified time interval, or as the result of a program decision. It should never be used as the last logical statement of a core load since the QUEUE routine returns control to the instruction immediately following the CALL QUEUE. A CALL ENDTS (see Use of Time-Sharing) statement is normally used in conjunction with CALL QUEUE for time-sharing systems. The main uses of CALL QUEUE can be summarized thus:

Problem: Repeated execution of queued core loads during a given core load.

Solution: (The encircled numbers specify the sequence of operations.)



Note 1: The CALL SPECL statements cause core load A to be saved before transferring to core load E via lines 3 and 8. The CALL BACK statement in core load R causes core load A to be restored before the return is made via lines 6 or 11.

Note 2: Between lines 4 and 5 all core loads of priorities 1 and 2 will be executed; between lines 9 and 10 all core loads of priorities 1 through 4 will be executed.

Figure 12. Illustrating a Method of Segmenting Mainlines Based on Scheduling Requirements

- To queue a core load from any program
- To queue a core load from any hardware operational level
- To queue a core load when the user is unaware what is presently in progress on any one machine level
- To queue a core load when the user is unaware what machine levels are in progress, and
- To queue a core load that is not related to all other core loads.

This is a very flexible command since related or unrelated core loads can be scheduled on the basis of time, a program decision, an interrupt, and from any hardware operational level.

CALL UNQ -- Delete from the Queue

The reverse of queueing a core load entry is to remove such an entry from the Queue Table in the system. The statement which gives this ability is:

CALL UNQ (NAME, P)

where

UNQ	=	Name of the subroutine that removes the specified mainline core load entry from the Queue Table
NAME	=	User-assigned name of mainline core load entry to be removed
P	=	Priority status of user-assigned core load NAME. This may be in the range 1-32767.

Upon execution of a CALL UNQ statement, control is transferred to the UNQ subroutine which searches the Queue Table for a similar entry of the same name and priority. If such an entry is detected, it is removed (that is, deleted) from the Queue Table. If the table does not contain a matching entry, the Queue Table remains unchanged. In either case, the UNQ subroutine returns control to the instruction immediately following the CALL UNQ statement. Like CALL QUEUE, CALL UNQ may be executed at any time and from any level of machine operation. Note that no error parameter is required.

CALL QIFON -- Queue Core Load if Indicator is On

The third queueing-type call is the CALL QIFON statement.

CALL QIFON (NAME, P, L, I, E)

where

NAME	=	User-assigned name of a mainline core load
P	=	Priority status of each NAME, in the range 1-32767.
E	=	Error parameter, as described for CALL QUEUE
L	=	Interrupt priority level indicator
I	=	PISW bit position indicator or CALL COUNT indicators

In TSX, a unique L and I combination parameter is set up for each process interrupt, program-settable interrupt, and program interval timer routine. The significance of this combination (which is dependent on the user's machine configuration) is given below:

L	I	Reference
0-23	0-15	Process interrupts
0-23	(-)n	Programmed interrupts (see CALL LEVEL)
(-)n	0-31	Subprogram number for CALL COUNT statements (see Interval Timers)

Minus (-)n above refers to any minus number.

The CALL QIFON function is required only when any of the above mentioned interrupts are set up to be recorded (for delayed servicing). In general, most interrupts call for immediate action, or as soon as their appropriate servicing program can be read from disk to variable core. Some interrupts, however, must be recognized immediately, but do not require action until a later time. The function of delaying servicing is termed "recording": the interrupt is then said to be "recorded". CALL QIFON thus provides the user with the ability to interrogate recorded interrupts only when he so desires. It is the only way a recorded interrupt can be serviced. Figure 13 illustrates the use of this function.

The core load entries are queued only if their respective interrupt record indicators are on. When an indicator is on, the QIFON routine sets up the

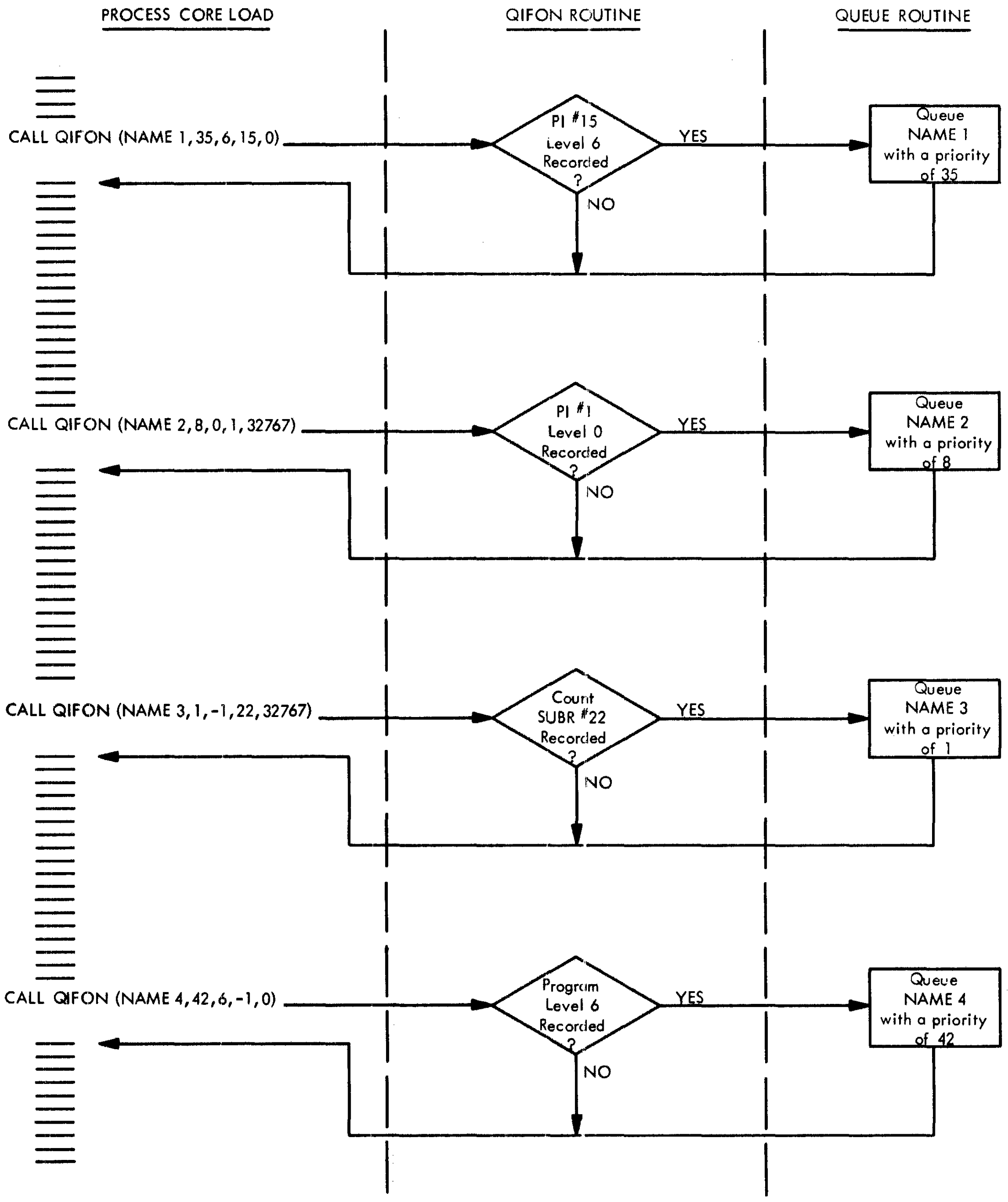

PROCESS CORE LOAD
QIFON ROUTINE
QUEUE ROUTINE
CALL QIFON (NAME 1,35,6,15,0)
PI #15 Level 6 Recorded ?
YES
NO
Queue NAME 1 with a priority of 35
CALL QIFON (NAME 2,8,0,1,32767)
PI #1 Level 0 Recorded ?
YES
NO
Queue NAME 2 with a priority of 8
CALL QIFON (NAME 3,1,-1,22,32767)
Count SUBR #22 Recorded ?
YES
NO
Queue NAME 3 with a priority of 1
CALL QIFON (NAME 4,42,6,-1,0)
Program Level 6 Recorded ?
YES
NO
Queue NAME 4 with a priority of 42

Figure 13. Use of the CALL QIFON Statement

proper information and then executes a CALL QUEUE. If the Queue Table is not full, or the replace error option is utilized, the QUEUE routine returns control to QIFON which proceeds with the interrogation of indicators until the QIFON call is completed. A recorded interrupt indicator is automatically turned off (that is, cleared) whenever the QIFON routine interrogates a program indicator. Control is then passed to the next executable instruction following the CALL QIFON statement, or as specified for error conditions under E.

CALL QIFON may be used from any level of machine operation. It should never be used as the last logical statement of a core load.

CALL VIAQ -- Execute Highest Priority Core Load

The fourth and last queueing statement is

CALL VIAQ

where

VIAQ	= Name of the subroutine that determines the highest priority core load entry in the Queue Table.

The CALL VIAQ statement, like CALL CHAIN, and CALL BACK, is used as the last logical statement of a core load. When executed, control is transferred to the VIAQ routine which interrogates the Queue Table. If the table is empty, the process is considered to be in an idle condition (that is, the process does not require any action at this time.) Since variable core is not utilized by process core loads, control is passed to the Time Sharing Control (TSC) program for nonprocess work if there is work to do. The Nonprocess Monitor indicates that it has batch work to perform by the execution of the Console Interrupt button, with sense switch 7 on. When the operator places a job stack in the card hopper, he turns on sense switch 7 and depresses the Console Interrupt button. This informs TSC that batch work is to be performed.

At the end of the job, the // END OF ALL JOBS card indicates no more batch work is to be performed until the Console Interrupt button is again depressed. This feature is provided to reduce the amount of disk activity, and to give faster response to the process whenever there is no nonprocess work for execution.

The time-sharing operation, thus initiated, will continue for the duration of time specified at system generation time, or until it is terminated by a CALL ENDTS statement. Note that a CALL VIAQ is automatically performed when time-sharing terminates. If, therefore, an interrupt program has previously placed a name in the queue, the named core load will then be immediately executed (see also Use of Time-Sharing). Figure 14 illustrates the use of this calling statement.

Problem: All programs of a given priority must be executed before a certain core load.

Solution:

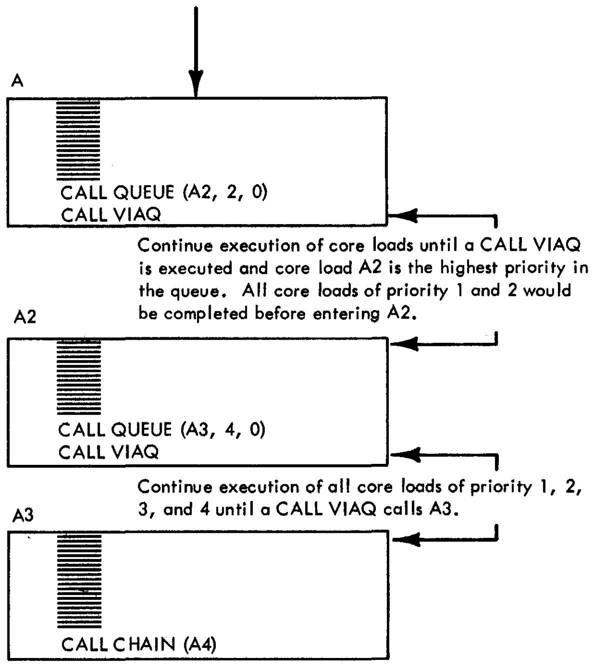

A
CALL QUEUE (A2, 2, 0)
CALL VIAQ
Continue execution of core loads until a CALL VIAQ is executed and core load A2 is the highest priority in the queue. All core loads of priority 1 and 2 would be completed before entering A2.
A2
CALL QUEUE (A3, 4, 0)
CALL VIAQ
Continue execution of all core loads of priority 1, 2, 3, and 4 until a CALL VIAQ calls A3.
A3
CALL CHAIN (A4)

Figure 14. Use of the CALL VIAQ Statement

In normal operations, the queue might not be empty, in which case the VIAQ routine obtains the name of the entry with the highest priority. If several entries have the same (highest) priority, the first entry of that priority will be selected.

The VIAQ routine then sets up the proper information for a CALL CHAIN with the core load name derived from the Queue Table, and passes control to PSC to execute the CHAIN function exactly as if a CALL CHAIN had been executed. Note that a core load containing a CALL CHAIN statement is destroyed by the core load it calls; a core load containing a CALL VIAQ is, therefore, similarly overlaid in core. The CALL VIAQ and CALL CHAIN commands are similar except for the method of obtaining the name of the core load to be called. Both calls, however, have their own useful unique functions.

Example of Non-synchronous Periodic Scheduling

The following example illustrates a simple technique frequently used in a process control environment whereby core loads can be executed on some periodic time basis. This is known as non-synchronous periodic scheduling. The test case is not intended as a model: it serves only to demonstrate program scheduling techniques. The example is given in three easy steps:

1. The Initial Core Load -- This is the initial mainline core load named TEST which is read into core by a cold start operation. The core load first unmasks the system because cold start enters the initial core load in an all-level masked condition; it then sets a programmed timer to initiate a continuous cycle of operations (by calling the count routine #0).

   Figure 15 illustrates this core load. The use of CALL CHAIN to call in another core load (that is, ALPHA) is also shown.
2. Mainline Core Load ALPHA -- This is the ALPHA core load called by the initial core load. It is a mainline core load which prints out the time of day (see Figure 16).

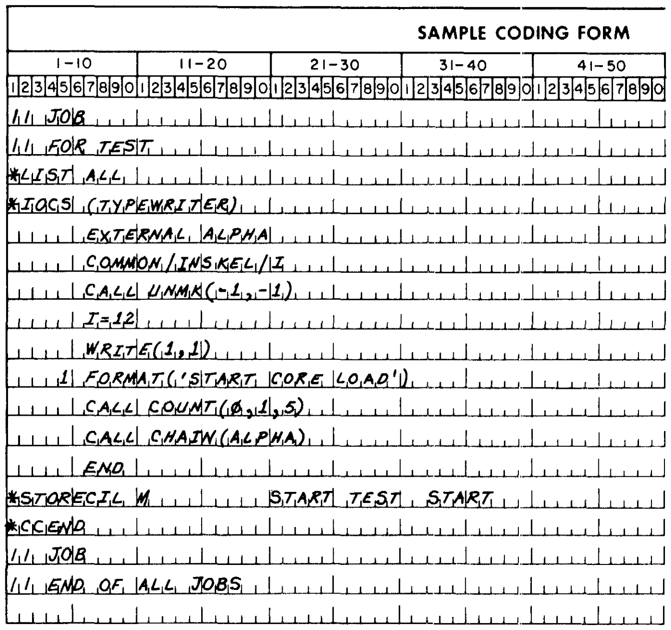
   Figure 16 also shows the use of CALL VIAQ to check the queue. If there is nothing in the queue, the system establishes the time-sharing mode (that is, the Nonprocess Monitor is called).

   If an // END OF ALL JOBS has just been executed, the VIAQ routine will wait until an interrupt occurs to check the queue.

   If time-sharing is in progress (that is, the Nonprocess Monitor is occupied), core is exchanged and the Nonprocess Supervisor is read into core, or alternatively, the interrupted nonprocess program is brought into core.
3. Count Routine PEROD -- This is the count routine named PEROD which is included in the System Skeleton at system generation time.

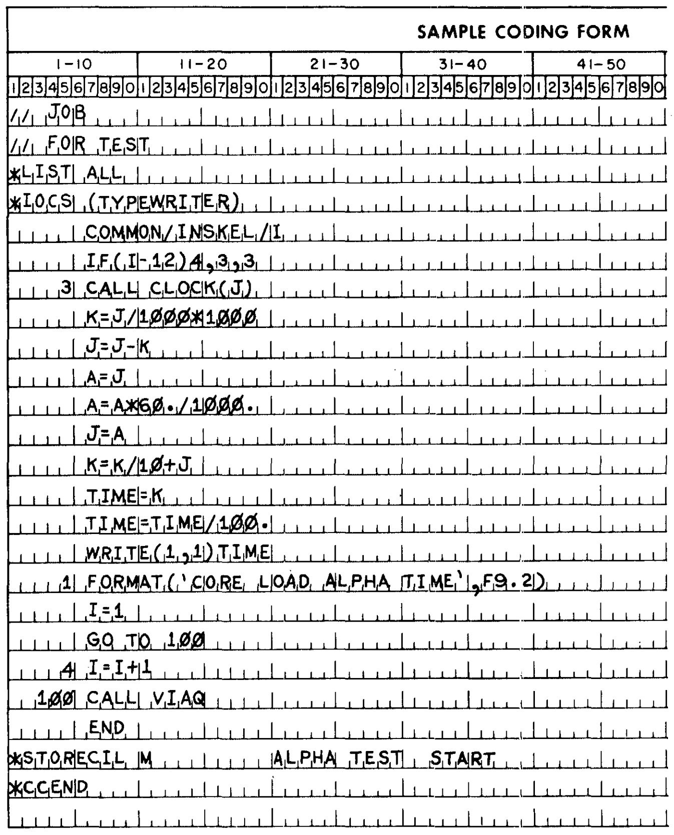
   It is entered by way of the Interval Timer Control (ITC) program when the time period specified in the initial core load TEST, or from its own call (that is CALL COUNT (0, 1, 5), has elapsed.

   The function of PEROD is to end time-sharing and to load ALPHA into the queue, so that when time-sharing is ended and the queue is checked, ALPHA will print out the time. It also restarts the timer to repeat this cycle of operation (that is, it starts the count again).


SAMPLE CODING FORM

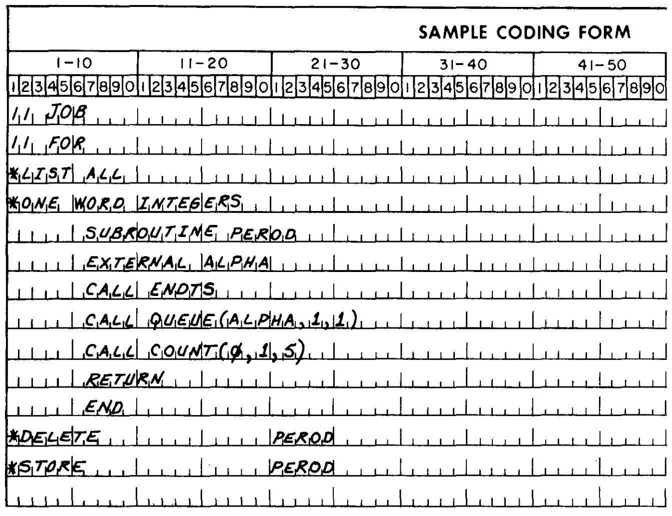
```
// JOB
// FOR TEST
*LIST ALL
*IOCS (TYPEWRITER)
      EXTERNAL ALPHA
      COMMON/INSKEL/I
      CALL UNMK(-1,-1)
      I=12
      WRITE(1,1)
    1 FORMAT('START CORE LOAD')
      CALL COUNT(Ø,1,5)
      CALL CHAIN(ALPHA)
      END
*STORECIL M         START TEST  START
*CCEND
// JOB
// END OF ALL JOBS
```

Figure 15. Initial Core Load


SAMPLE CODING FORM

```
// JOB
// FOR TEST
*LIST ALL
*IOCS (TYPEWRITER)
      COMMON/INSKEL/I
      IF(I-12)4,3,3
    3 CALL CLOCK(J)
      K=J/1ØØØ*1ØØØ
      J=J-K
      A=J
      A=A*6Ø./1ØØØ.
      J=A
      K=K/1Ø+J
      TIME=K
      TIME=TIME/1ØØ.
      WRITE(1,1)TIME
    1 FORMAT('CORE LOAD ALPHA TIME',F9.2)
      I=1
      GO TO 1ØØ
    4 I=I+1
  1ØØ CALL VIAQ
      END
*STORECIL M         ALPHA TEST  START
*CCEND
```

Figure 16. Mainline Core Load ALPHA



SAMPLE CODING FORM

```
// JOB
// FOR
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE PEROD
      EXTERNAL ALPHA
      CALL ENDTS
      CALL QUEUE(ALPHA,1,1)
      CALL COUNT(Ø,1,5)
      RETURN
      END
*DELETE             PEROD
*STORE              PEROD
```

Figure 17. Count Routine PEROD

HANDLING OF INTERRUPTS

Interrupt Philosophy

Basically, in all on-line real time control systems, the processor-controller behaves in very much the same fashion as a radar system. The real-time computer reacts to input data from a real world environment and provides input data to correct or control that environment. For example, a computer system controlling a chemical process monitors the inputs from measuring devices and instrumentation on the operator's control panel. Later, the computer updates the control mechanisms and indicators to maintain safe and efficient operation. Emergency conditions are also sensed and appropriate action initiated. Instrument status sensing, data computation, and reaction control must occur within a specified interval of time to prevent disruption of the process. How well it is able to respond generally determines the maximum capability of the on-line system. A significant component in the responsive ability of any real time system is the inclusion of a powerful and flexible multi-priority interrupt program.

Purpose of I/O Interrupts. There are two main reasons for I/O interrupts:

1. To reduce system cost by reducing control circuitry in I/O devices
2. To speed up job throughput, which is relatively slow when compared with internal processing.

Consider a normal computer operation without interrupts. Since the computer is basically a sequential machine, it functions sequentially (or serially, performing one job at a time). In the simple example below,

INPUT1 - PROCESS1 - OUTPUT1 - INPUT2 - PROCESS2

when PROCESS1 is completed, the user must wait until OUTPUT1 and INPUT2 are accomplished before he can begin PROCESS2. This could be extremely time-consuming.

Since the input device waits idly during PROCESS1 and OUTPUT1 time, the question arises: why should this idle interval of time not be used to read in INPUT2? This could be obviated with the use of I/O interrupts. The I/O interrupt is based on the concept of keeping I/O devices active, thus, hopefully, eliminating process delay caused by these devices. The following sequence of events illustrates the type of action that might be taken:

1. A mainline program initiates an I/O device operation.
2. The program proceeds with its processing while the I/O device is sending (or receiving) information.
3. When the I/O device has transferred its information, an interrupt signal is sent to the Process Controller.
4. This interrupts the mainline program.
5. The interruption is serviced; that is, further data is requested or sent.
6. The mainline resumes processing at the point of interruption.
7. The cycle repeats itself during the execution of the program.

1800 Multi-Interrupt Priority Scheme. In the IBM 1800 Time-Sharing Executive System, the essential elements of the multi-interrupt priority control scheme consist of:

- A hardware priority structure
- Core store data areas for each interrupt level
- A Master Interrupt Control Program (MIC) which recognizes, controls, and directs the servicing of interrupts

The hardware priority structure provides for 3 fixed and up to 24 additional interrupt levels which are assignable by the user to I/O, process, or programmed interrupts, as shown in Figure 18.

The interrupt philosophy can be explained in the following way. Because of the large number and widely varying types of interrupt requests, it is often not desirable to cause a branch to a unique address for each condition. For the same reasons, it is not desirable to initiate one branch for all interrupt requests and to require the program to determine the individual requests requiring service. Grouping the numerous request lines into a number of priority levels (see Figure 18) accomplishes two aims:

1. It allows all interrupt requests common to a specific interrupt level to have the privilege of interrupting immediately, if the only requests present are of a lower priority level. Conversely, it permits interrupt requests connected to a higher priority level to temporarily terminate the servicing on a lower level and to immediately interrupt to the higher priority level. Service is returned to the initial request only after all higher level requests have been serviced.

INTERRUPT			PRIORITY LEVEL ①	DECIMAL ADDRESS	ILSW	PISW ② ASSIGN'T	MASK & UNMASK	PROGRAM INTERRUPT	I/O, TIMER, PROCESS INTERRUPT: ASSIGNMENT ALLOWED
	Internal		1	8	Yes	–	No	No	No
	Trace		26	9	No	–	③	No	No
	CE		27	1 ④	No	–	No	No	No
	Assigned	0	2	11	Yes	2	Yes	Yes	Yes
	Levels	1	3	12	Yes	3	Yes	Yes	Yes
		2	4	13	Yes	4	Yes	Yes	Yes
		3	5	14	Yes	5	Yes	Yes	Yes
BASIC		4	6	15	Yes	6	Yes	Yes	Yes
		5	7	16	Yes	7	Yes	Yes	Yes
		6	8	17	Yes	8	Yes	Yes	Yes
		7	9	18	Yes	9	Yes	Yes	Yes
		8	10	19	Yes	10	Yes	Yes	Yes
		9	11	20	Yes	11	Yes	Yes	Yes
		10	12	21	Yes	12	Yes	Yes	Yes
		11	13	22	Yes	13	Yes	Yes	Yes
		12	14	23	Yes	14	Yes	Yes	Yes
SPECIAL FEATURE GROUP 1		13	15	24	Yes	15	Yes	Yes	Yes
		14	16	25	Yes	16	Yes	Yes	Yes
		15	17	26	Yes	17	Yes	Yes	Yes
		16	18	27	Yes	18	Yes	Yes	Yes
		17	19	28	Yes	19	Yes	Yes	Yes
		18	20	29	Yes	20	Yes	Yes	Yes
SPECIAL FEATURE GROUP 2		19	21	30	Yes	21	Yes	Yes	Yes
		20	22	31	Yes	22	Yes	Yes	Yes
		21	23	32	Yes	23	Yes	Yes	Yes
		22	24	33	Yes	24	Yes	Yes	Yes
		23	25	34	Yes	25	Yes	Yes	Yes

① NOTE: 1 Highest priority
27 Lowest priority
② 24 PISW's Basic IBM 1800.
③ Manually masked and unmasked by switch.
④ Return address in I-counter stored in decimal address 0010, but hardware-generated BSI addresses decimal address 0001.

Figure 18. Priority Interrupt Level Structure and Assignment

2. Since a unique branch can be defined for each interrupt priority level, it is possible to combine many requests on a common priority level and thereby use a common interrupt subroutine to service many requests.

Each interrupt request line is thus positioned into a table order of priority; the highest priority being closest to the output, while the lowest priority is farthest away. An interrupt request received at a given level automatically causes the level to shift from an uninterrupted to an interrupted state. If no higher priority level is presently being served, the scheme permits the request line to be activated. At this time, a unique address associated with this level is supplied to the system, which transfers control to a core location determined by this address. The mainline return address is now preserved and entry made to the Master Interrupt Control Program to direct the servicing of this interrupt. At completion of servicing, control is returned to the point of departure (see Figure 9).

In this way, every interrupt request is obeyed immediately, provided no priority request is presently in execution. The biggest advantage of this method of priority level control is a near-optimum priority response. To guarantee minimum response time to alarm conditions, most process interrupt servicing routines should be in core at all times.

Characteristics of Interrupts

Interrupts can be classified into three broad types:

- I/O
- External (that is, process), and
- Programmed

Skeleton-resident interrupts operate on a true priority basis from the 24 levels available. An interrupt is, by definition, a hardware feature -- it is the machine hardware, not the Master Interrupt Control Program which determines what level the interrupt is on. As far as the problem programmer is concerned, he has no control over the time of occurrence of process interrupts. He has, however, indirect control of their time response through masking, recording, and the allocation of priority levels. In general, interrupts are distinguishable from one another only in the manner in which they are serviced (see also System Design Considerations).

Priority Assignments. Some important considerations affecting priority assignments can be summarized thus:

- Each of the 24 levels can interrupt the mainline program.
- Level 0 is the highest priority.
- Higher priority levels can interrupt lower priority levels. Lower priority levels cannot interrupt higher levels. This permits fast access devices to interrupt slower ones.
- Hierarchy of machine operation:

Highest	Interrupt level	0
		1
		2
		●
		●
		●
		●
		●
		23
	Process Mainline	
Lowest	Nonprocess Mainline	

- Interrupt levels may be masked by programming means. Masking inhibits interrupts to the 1800. The user is thus allowed to inhibit or permit specified levels of interrupts, and to allow determination of the status of interrupt levels -- that is, whether they are inhibited or not -- at any time. Through selective use of masking, data channels can be kept in operation for the transmission of data into and out of core storage while process interrupts are prevented from occurring. This gives an increased efficiency of execution of programs.

  When a request line is unmasked, the Processor-Controller is interruptible. Note that although a level may be masked, the fact that the interrupt has occurred is not lost. The function of masking is used to delay recognition of an interrupt.

In practice, priorities must be assigned using the interaction of functions with each other as a primary basis. See also System Design Considerations: System Director.

Types of Servicing Subroutines Used

An interrupt servicing subroutine may be

- An I/O device subroutine
- An interrupt program included in the skeleton
- An interrupt program included with a mainline
- A mainline core load
- An interrupt core load

The different options are provided to permit flexibility in terms of both core storage and response time requirements.

I/O Device Subroutines. An I/O device routine is a routine that performs the second level of sensing of a Device Status Word (DSW) or a Process Interrupt Status Word (PISW). The first level of sensing the Interrupt Level Status Word (ILSW) is carried out by MIC. This means that any bit on the ILSW that requires sensing at the second level may be executed by an I/O device routine.

The majority of the I/O devices in the 1800 have IBM-supplied device routines (e.g., disk, card/read punch). Those that require sensing by the user at the second level include the following:

- RPQ devices
- Special PISW's that the user may wish to sense himself (e.g., multiple PISW groups per level)
- Any other I/O device (e.g., System/360 Channel Adaptor)

These routines are entered with a BSC; they exit by an indirect branch through word $(90)_{10}$.

The appropriate entry reflecting the ILSW will be assigned by the user on *Assignment control cards to the System Loader at system generation time.

Subroutines that are entered from the I/O device routine comprise count, timer, and process I/O subroutines. They perform specific limited tasks associated with the event that is occurring within the I/O device (e.g., elapsed time on a particular timer). Entry to the subroutine is made by a BSI; the routine exits to its return statement by a BSC I through the entry point. These subroutines are included in the skeleton by *INCLD control cards.

Interrupt Programs included in the Skeleton. The shortest response time (that is, the minimum time before an interrupt servicing routine is entered after the interrupt has been recognized) is obtained by placing the routine in core with the System Skeleton. The interrupt routine is included by specifying a control card (*INCLD) at skeleton build time. Like the interrupt core load, the in-core interrupt routine performs a limited task. It is masked only for short periods of time by the system during the execution of certain reentrant coded routines. This period of time is normally of the order of 20-30 instructions.

These routines are entered with a BSI; they exit through a CALL INTEX statement. Some of the important factors governing their inclusion in the skeleton area are discussed in detail in System Design Considerations: System Skeleton.

Interrupt Programs included with a Mainline. Next in length of response time to skeleton interrupt routines are in-core routines loaded with the core image mainlines. These are entered almost as quickly as skeleton routines provided the mainline is in progress when the interrupt occurs, but may be forced to wait if the mainline is not in core. This will be the case if a lower level interrupt routine has been read over the mainline. The length of delay involved would then be the balance of the reading of the interrupt routine and the execution of that routine and the read-back of the mainline. No immediate exchange to obtain the mainline is done. If the interrupt that occurs has a program in the mainline and the interrupt is at a higher or equal level to the interrupt being processed, the interrupt core load assigned to this interrupt will be read directly into core upon completion of the interrupt core load being serviced.

An interrupt core load is always required before any servicing of a process interrupt in-core with the mainline can take place. If an interrupt core load is not available, the event will be recorded even if an interrupt servicing route is included with the mainline. The Master Interrupt Control (MIC) Program first ascertains if an interrupt core load is available; if it is, the ICL table is checked to see if the routine is in the mainline; if it is not available, the event is recorded. The interrupt routine is always serviced with the same masked status as an interrupt core load.

Interrupt programs included with a mainline are always entered by an indirect branch (BSI); they exit through a CALL INTEX.

Mainline Core Loads. External (that is, process) interrupts whose occurrences are recorded are serviced with mainline core loads. The mainline core load performing the servicing action is identical to any other mainline core load, except that it is queued for execution by a CALL QIFON statement. Since it is a queued core load, it should have a CALL VIAQ as its last logical statement. It could, of course, be the first core load of a special series, in which case it would end with a CALL CHAIN to obtain the next core load in sequence, but a CALL VIAQ must ultimately be executed.

Note that the only major difference between an interrupt core load and a mainline core load used for the servicing of recorded interrupts is in the last logical statement used. This must be a CALL INTEX for an interrupt core load and a CALL VIAQ for a mainline core load.

If a process interrupt is immediately serviced on some occasions and recorded on other occasions, it would require two core loads (one for each function) which would be identical in all respects except for their last logical statement. To eliminate this duplication of core loads, a special combination exit statement (CALL DPART) is provided (see Exit Procedures from Interrupt Servicing Routines). An interrupt or mainline core load which terminates with a CALL DPART is, by definition, a combination core load.

The combination core load should not violate restrictions placed on either mainline or interrupt core loads. That is, mainline interrupt subroutines are not allowed as part of this core load: only statements allowed in both mainline and interrupt programs are permitted. See also Appendix A, Summary of TSX Statements.

Interrupt Core Loads. The user may create interrupt core loads which are brought into core over the mainline when the interrupt occurs. Interrupt core loads are essentially disk-resident routines where the response time is not a problem. They are required for those interrupts that meet either of the following conditions:

1. The user has specified the interrupt servicing routine to be out-of-core.
2. The user has specified the interrupt servicing routine to be in-core as part of a mainline core load.

When this type of interrupt servicing routine is executed, the area of core that the routine will occupy is saved on disk before reading in the interrupt core load. The time for this save operation, in addition to the time for the disk read operation needed to get the interrupt core load, causes this method of interrupt servicing to have the longest response time. Once an interrupt servicing core load has begun, it may be interrupted by a higher level routine, only if the interrupt routine for this higher level is in the skeleton on a higher level.

The use of interrupt core loads is normally restricted to the performance of a particular task at a time, or the initiation of a task on a mainline level which does not take an inordinate amount of time. A typical example is the queueing of a sequence of mainline core loads to accomplish the task that originated an interrupt. The user should remember that if his problem program is time-consuming, he will, in the normal course of events, execute this on the mainline level. The reason for this is that interrupt core loads cannot, by definition, interrupt other interrupt core loads. This system restriction is because of the disk exchange time that would be required.

Interrupt core loads are built and assigned to a particular process interrupt bit (PISW) on programmed interrupt level. The core load then performs the servicing task or sets in motion the task that will be required when this specific bit is activated.

Note that this type of interrupt servicing routine does not contain an Interrupt Status Table (IST). The reason is that the IST is used for updating the Interrupt Core Load Table (ICLT), and the ICL table is only updated from mainline core loads or from combination core loads when these are executed at the mainline level. For the same reason, the interrupt core load cannot include other routines within it. Another explanation is that the programs that might be included in an interrupt core load are masked off during its execution.

An interrupt core load is not necessarily the length of variable core: it has a defined length (see System Design Considerations: Disk System Configuration). Hence, in contradistinction to mainline core loads, all of variable core is not needed because of the limited function performed by this type of core load. To increase execution speed, therefore, the Interrupt Save Area can be made smaller.

Note also that interrupt core loads can communicate with mainline core loads (and combination core loads when these are executed as interrupt core loads) only through INSKEL COMMON. The interrupt core load itself contains a COMMON which is located at the end of the Interrupt Save Area.

Figure 19 gives a summary of the types, characteristics, and location of process interrupt servicing routines.

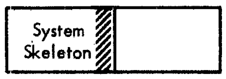
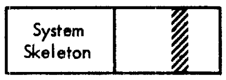
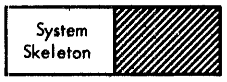
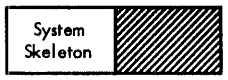
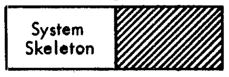
Type of Routine and Location	Characteristics
Skeleton Interrupt Routine Core Storage Location  Skeleton Area Variable Area	Permanently in core. Normally high priority. Can immediately interrupt lower priority routines, and Interrupt Core loads if no Interrupt Core load is assigned to that level. Fastest interrupt response. Must CALL INTEX as last logical statement.
Mainline Interrupt Routine Core Storage Location  Skeleton Area Variable Area	Available almost as quickly as Skeleton Interrupt routines, if the mainline is in-core. Once execution is started, only interruptable by Skeleton Interrupt Routine or internal interrupt. Can be different with each mainline core load. Interrupt core load is required. Must CALL INTEX as last logical statement.
Interrupt Core Load Core Storage Location  Skeleton Area Variable Area	Large core area available. Once execution is started, only interruptable by Skeleton Interrupt Routine or internal interrupt. Mainline or nonprocess program in operation at time of interrupt is saved before and restored after Interrupt Core Load operation. CALL INTEX is last logical statement used. Cannot include interrupt routines for other interrupts.
Mainline Core Load Core Storage Location  Skeleton Area Variable Area	Large core area available. Can include interrupt routines. Queued for execution if record indicator is on when named in QIFON statement. If mainline core load is always queued, last logical statement should be CALL VIAQ.
Combination Core Load Core Storage Location  Skeleton Area Variable Area	Cannot violate any rules governing interrupt and mainline core loads. Large core area available. Queued for execution if record indicator is on when named in QIFON statement. CALL DPART is last logical statement used.

Figure 19. Summary of Characteristics of Process Interrupt Servicing Routines

Exit Procedures from Interrupt Servicing Routines

Three forms of exiting are used:

- CALL INTEX
- RETURN
- CALL DPART

CALL INTEX -- Interrupt Exit

All interrupt routines serviced on an interrupt level must return control to MIC through a

CALL INTEX

statement. INTEX is the symbol for INTerrupt EXit. CALL INTEX must be used as the last logical Statement in skeleton interrupt routines. It can also be used in interrupt core loads.

RETURN

Subprograms called by user-written interrupt servicing routines must use a

RETURN

statement to return to the interrupt routine or may return control directly to MIC.

CALL DPART -- Departure

CALL DPART causes the level of operation to be tested for the following conditions:

- If the present level is an interrupt level, a CALL INTEX is executed.
- Otherwise a CALL VIAQ is executed.

Thus CALL DPART eliminates duplication of core loads. An interrupt that is sometimes directly serviced, and sometimes recorded, can now be serviced with the same core load. This core load operates from an interrupt level when servicing is specified; it is queued and operates from the mainline level when the interrupt is specified as recorded.

Figure 20 illustrates the use of the two exit CALL and RETURN Statements.

Master Interrupt Control

Once an interrupt has been detected at the hardware level, a reentrant control program, the Master Interrupt Control (MIC) program, takes over the control and servicing of that interrupt. The interrupt is first recognized by the interrogation of certain indicators on a level.

The MIC routine is assembled as part of the System Director at which time it origins out those tables and coding not used by the system to user specifications. MIC resides in core at all times in an on-line TSX system when the computer is operating under control of the System Skeleton. It is designed to:

- Save the interrupted registers whenever an interrupt is processed on the appropriate work level
- Direct the interrupt to its servicing routine
- Restore the FORTRAN I/O buffers if required
- Restore the interrupted registers, and
- Return to the point of departure in the interrupted program.

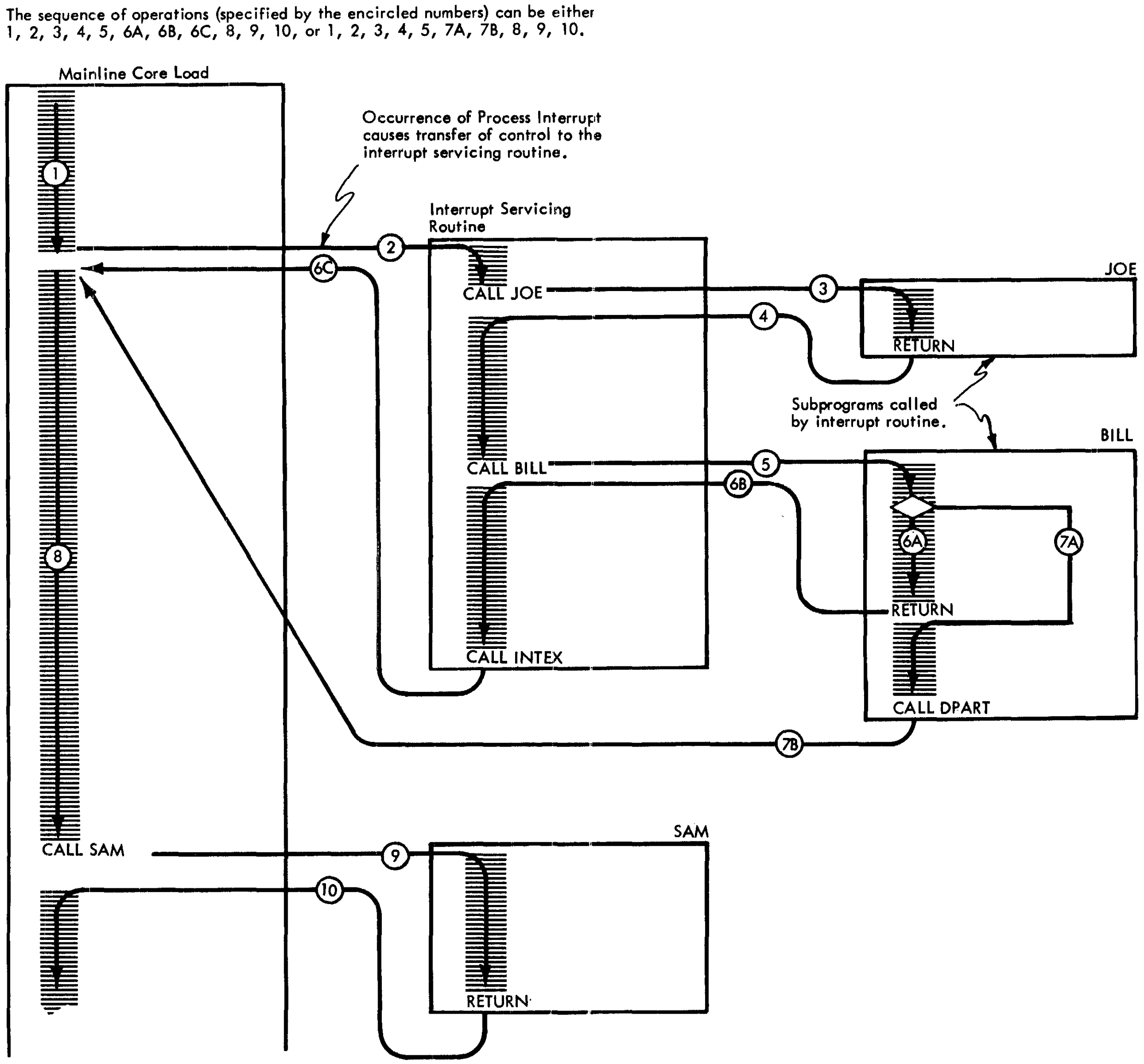
Detailed Action of MIC when an Interrupt Occurs

Consider the train of events that follows when a process interrupt is generated by an event within a process control environment. Let us assume that this interrupt was originally assigned (at system generation time) by the user on an NB (System Director) equate card to level zero. Remember that an interrupt is, by definition, a hardware feature, and that the user has limited control over the time of occurrence of process interrupts, except by masking, recording, and the allocation of priority levels. Figures 21, 22 and 23 illustrate this action in simplified form.

Entry to MIC.

1. In the 1800, an interrupt request is recognized at the completion of the current instruction

being executed within a mainline program. When this happens, an indirect branch (BSI) to a fixed-word (location 11) in core takes place. This word contains the start address of a level work area associated with level 0 (see System Design Considerations: System Director). A set of instructions within this area then sets the level busy, saves Index Registers 1, 2 and 3, and sets Index Register 3 as a pointer to this work level (at entry point + 8). It is through the level work area that an interrupt formally enters MIC -- from now on, all references to the work area and saved information is made through the Index Register 3 address.



The sequence of operations (specified by the encircled numbers) can be either 1, 2, 3, 4, 5, 6A, 6B, 6C, 8, 9, 10, or 1, 2, 3, 4, 5, 7A, 7B, 8, 9, 10.
Mainline Core Load
Occurrence of Process Interrupt causes transfer of control to the interrupt servicing routine.
Interrupt Servicing Routine
1
2
6C
CALL JOE
3
JOE
4
RETURN
Subprograms called by interrupt routine.
BILL
CALL BILL
5
6B
6A
7A
RETURN
CALL INTEX
CALL DPART
7B
8
CALL SAM
9
SAM
10
RETURN

Figure 20. Use of the CALL INTEX, CALL DPART, and RETURN Statements.

2. MIC is the entry point at which all process (and I/O) interrupts enter the Master Control Program for processing. The accumulator, the status word, and the pseudo-accumulator, are now saved for the particular level of interrupt being processed. The previous (that is, last) work level address is also saved, and the new (that is, current) work level address set up for use by reentrant coded subroutines so that they are aware of the address of the particular work level they are required to use at this particular time. Now that the registers of the interrupted level have been saved and the new level (0) address set up, the question of determining which of 16 possible interrupts is to be serviced on this level remains. This is done by sensing the ILSW. If no bits are "on" in the ILSW, a check is made to see whether a programmed interrupt has been selected for this level; if it has been, a transfer is made to (A) in Figure 22, and the processing procedure proceeds as for a process interrupt. If no programmed interrupt is present, an exit from MIC is made via (B) -- see Figure 23.
3. If a bit is on, a branch is made via the level work area to the Interrupt Branch Table within the mainline core load to determine whether the interrupt is a process or I/O interrupt.

   Each core load (mainline, combination, interrupt, or nonprocess) must contain an Interrupt Branch Table which provides the means of routing each I/O, process, or programmed interrupt to its appropriate servicing routine. The table, built in reverse order as shown in Figure 21, consists of single-word entries, each of which contains either an entry address to an I/O device servicing routine for an I/O interrupt, or a fixed address within the Skeleton for a process interrupt. The table is initially built by the Skeleton Builder and Core Load Builder to the specifications of the System Loader. Its size is determined by the number of bits on all interrupt levels used.

   Since we are concerned with a process interrupt (LEVEL BIT 0 = PISW, see Figure 21) level 0 will contain the entry point PRIE (that is, the reentry point to MIC). (Note that if an I/O interrupt were present instead, the I/O servicing routine is entered. The case of an I/O interrupt occurrence is discussed later).
4. The PISW derived from the work level is now sensed. If no bits are on (that is, no event has taken place within the process control environment) the exit route (from MIC) is taken via (B). If a bit is on, it is reset, and the address of the ICL table associated with this particular interrupt set up.
5. Now that the process interrupt is correctly known, the option of processing must be interrogated and executed -- that is, we must now determine what type of servicing this particular process interrupt requires. Various tests are performed to determine:

   - Whether the interrupt is to be recorded
   - Whether the interrupt servicing routine is in core with the skeleton
   - Whether the interrupt is to be serviced by an out-of-core interrupt core load, or
   - Whether the interrupt servicing routine is in core with the mainline

   in conjunction with entries made in the ICL Table (see <u>System Design Considerations: System Director</u>).

   The first test ascertains whether this particular interrupt is to be recorded. If it is, a subroutine records the interrupt. If it is not to be recorded, a check is made to see if the interrupt servicing routine is included with the skeleton. If it is, it is serviced by that subroutine. The next test determines whether an interrupt core load has been loaded to the disk to service this interrupt. If it has not, the interrupt is automatically recorded. If it has, all interrupt levels serviced by out-of-core routines will be masked. This also prevents a user from unmasking any level that is asssociated with out-of-core interrupts.

   A test is now made to determine if the interrupt servicing routine is in core with the mainline program. If it is in core with the mainline, the mainline itself is in core, and we are not in an exchange of variable core; the Index Register is then set to the transfer vector, and the entry point of the interrupt servicing routine is located in the Interrupt Status Table. Entry points to interrupts in core with the mainline are situated in a table known as the Interrupt Status Table (IST). The format of the table consists of:

   - One word indicating the length of the table for each level
   - One word for interrupts that are in core with the mainline

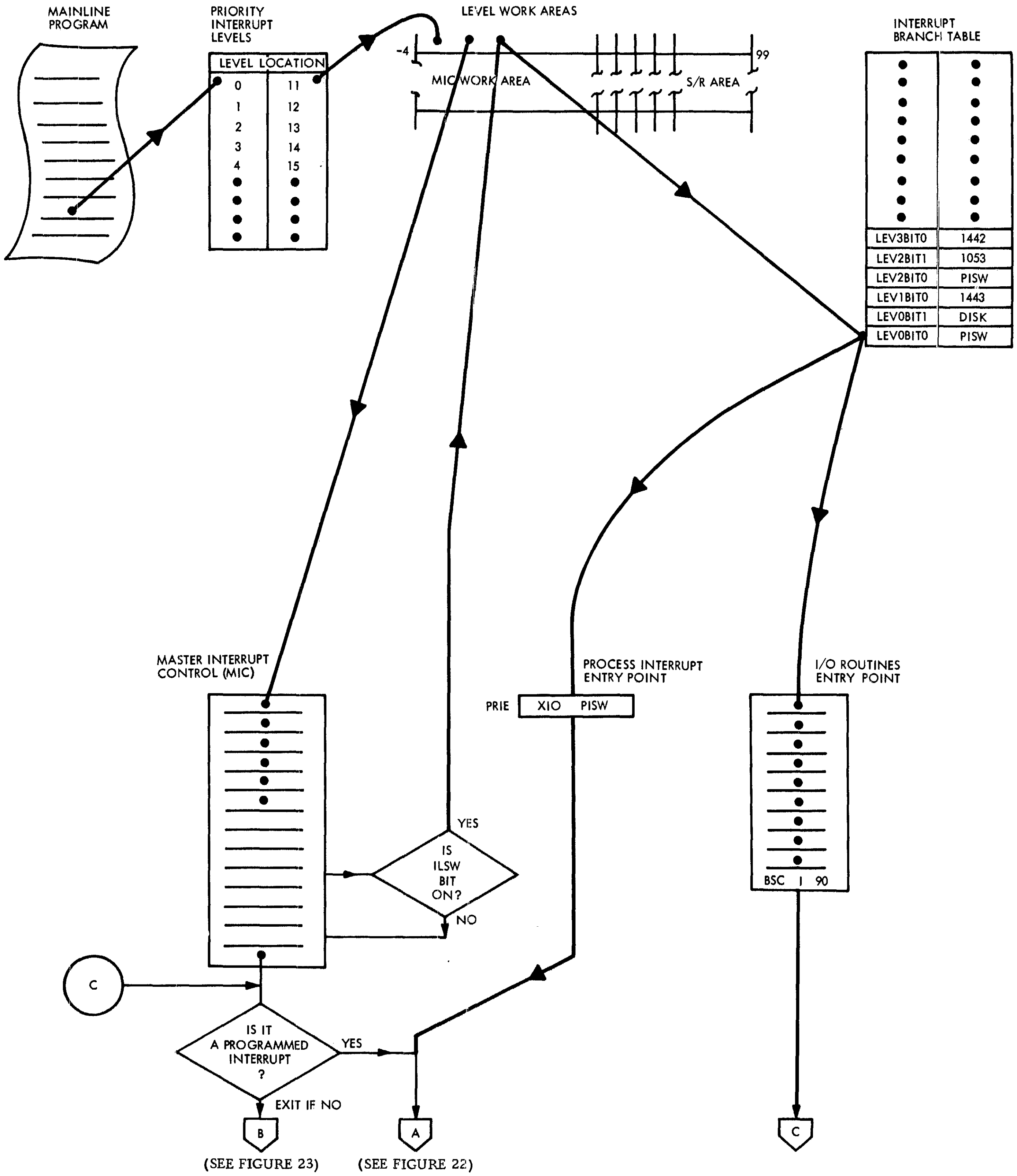


Figure 21. Action of MIC During an Interrupt

- One word for interrupts which are to be recorded on a particular level

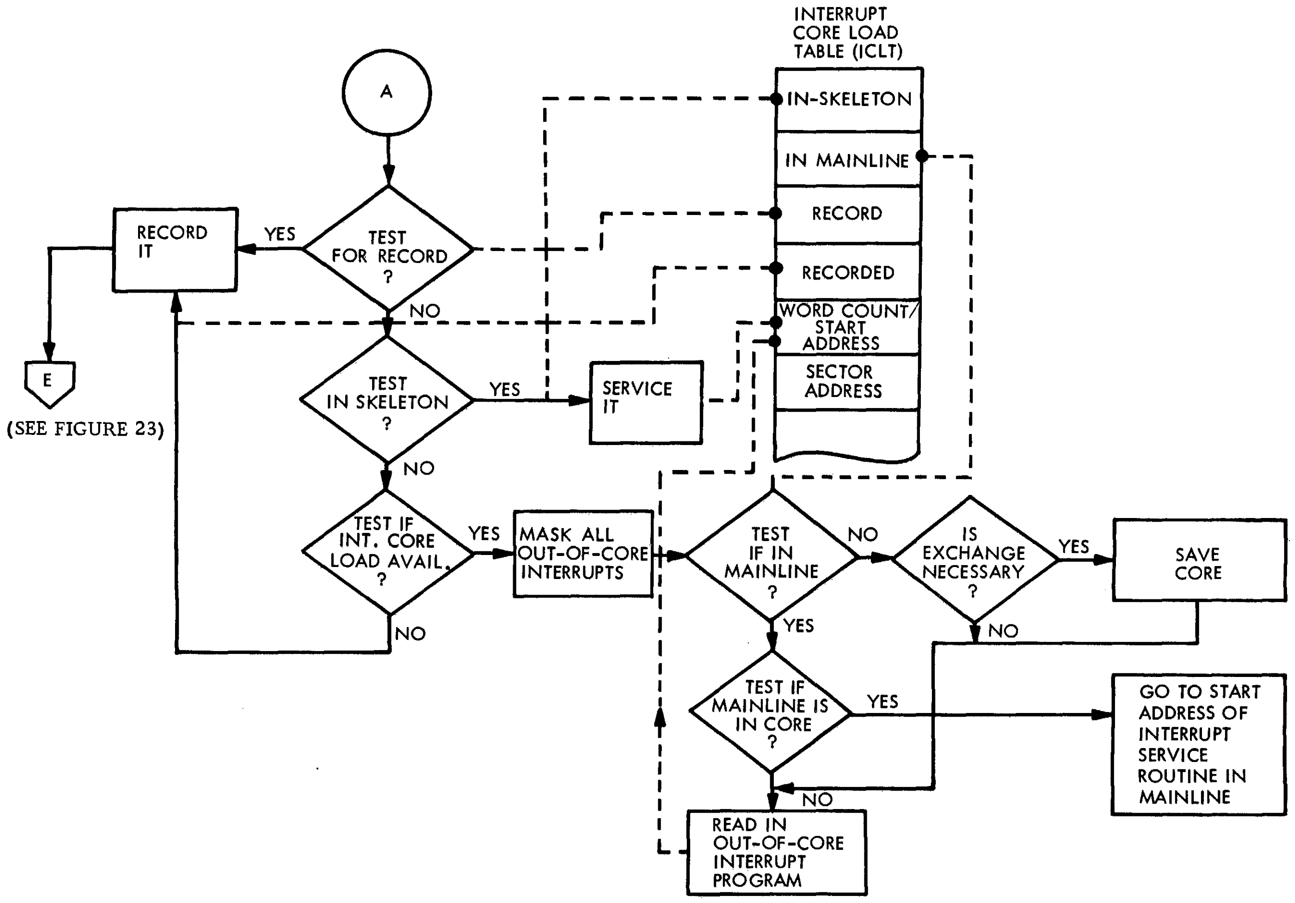
followed by as many words as are necessary to contain the start address of interrupts in core with the mainline. The size of the table is determined by the user when he defined his system.

If the interrupt is an out-of-core interrupt, I/O must be completed in the mainline area prior to either exchanging core, or, if we are in an exchange, prior to reading in the interrupt core load. Once the interrupt core load is read into core, Index Register 3 is set to the transfer vector and the interrupt entered for execution. An exchange means that variable core has been saved in the Interrupt Save Area on disk. The area exchanged will be the size of the largest interrupt program specified by the user.

Note that due to cycle stealing I/O, some area may be either modified or recorded at the time the process interrupt occurred. This means that out-of-core interrupts must always be assigned to a priority level lower than all I/O devices.

Exit from MIC. All process interrupt programs terminate by a return CALL INTEX statement to MIC. INTEX is the address to which interrupt servicing programs return upon completion of their processing. An exit procedure is now made to either of two routes dependent on the type of servicing routine just executed. That is, whether the servicing routine was an in-core-with-Skeleton routine or an out-of-core servicing routine.

If it is in core with the skeleton, and this is the last servicing required (no further PISW bits on), a common exit from MIC is taken via (B) and (C) -- see Figures 21 and 23. Note that this is also the



INTERRUPT CORE LOAD TABLE (ICLT)
IN-SKELETON
IN MAINLINE
RECORD
RECORDED
WORD COUNT/ START ADDRESS
SECTOR ADDRESS
A
TEST FOR RECORD ?
YES
RECORD IT
NO
E
(SEE FIGURE 23)
TEST IN SKELETON ?
YES
SERVICE IT
NO
TEST IF INT. CORE LOAD AVAIL. ?
YES
MASK ALL OUT-OF-CORE INTERRUPTS
NO
TEST IF IN MAINLINE ?
NO
IS EXCHANGE NECESSARY ?
YES
SAVE CORE
NO
YES
TEST IF MAINLINE IS IN CORE ?
YES
GO TO START ADDRESS OF INTERRUPT SERVICE ROUTINE IN MAINLINE
NO
READ IN OUT-OF-CORE INTERRUPT PROGRAM

Figure 22. Action of MIC during an Interrupt (Continued)

exit point for all I/O interrupt routines. If additional process interrupts are indicated (that is, more bits for PISW sensing are on) the exit route proceeds to (A) -- see Figure 22 -- and the procedure continues in the normal fashion of a process interrupt recognition. A closed loop is thus maintained until all process interrupts have been serviced, finally exiting through the common exit point (B) for all categories of interrupts.

If the return originated from the servicing of an out-of-core interrupt program, all out-of-core interrupt levels are unmasked at this point to allow other out-of-core interrupts to occur, so that it is not necessary to carry out an exchange of variable core for the servicing of that particular interrupt. Because the unmask instruction masked out all levels for one more instruction, the branch out or exit can be executed prior to any interrupts occurring.

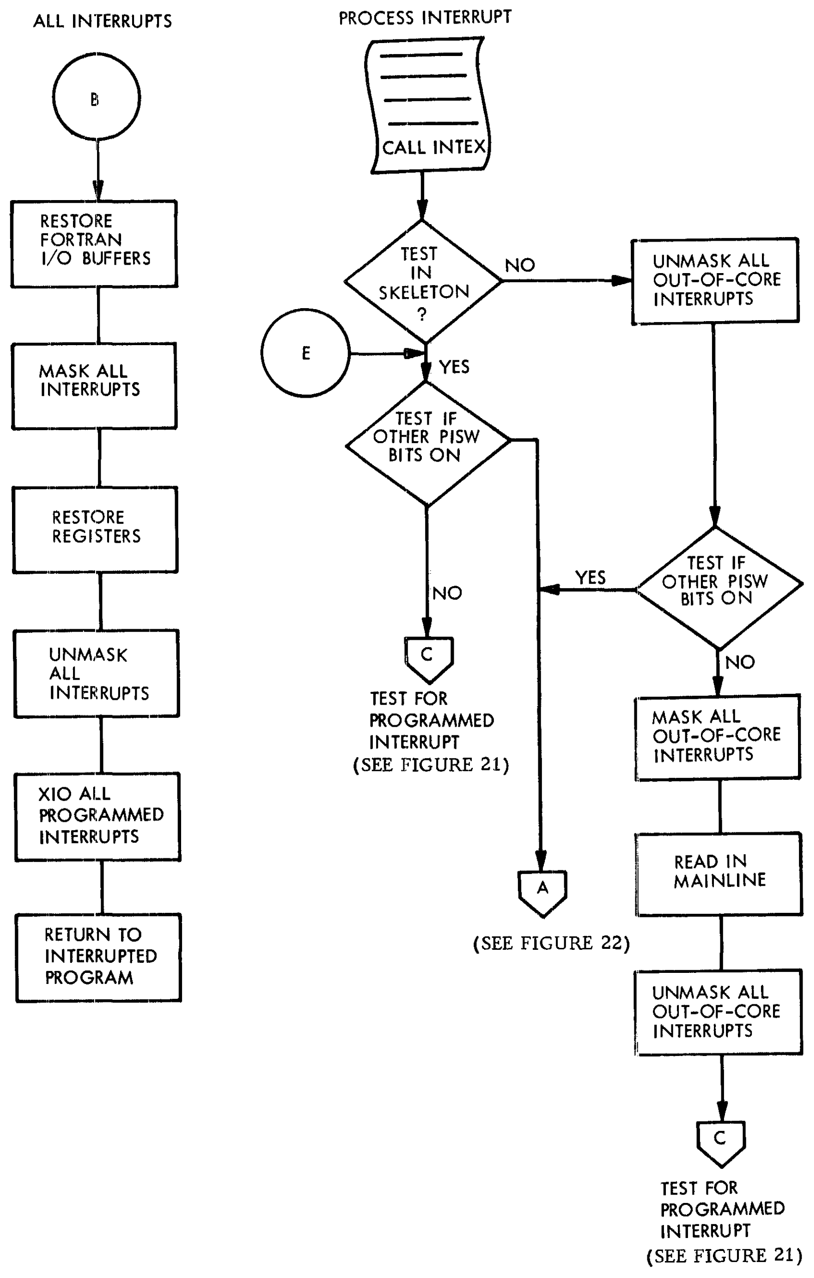

ALL INTERRUPTS
B
RESTORE FORTRAN I/O BUFFERS
MASK ALL INTERRUPTS
RESTORE REGISTERS
UNMASK ALL INTERRUPTS
XIO ALL PROGRAMMED INTERRUPTS
RETURN TO INTERRUPTED PROGRAM
PROCESS INTERRUPT
CALL INTEX
TEST IN SKELETON ?
NO
YES
E
UNMASK ALL OUT-OF-CORE INTERRUPTS
TEST IF OTHER PISW BITS ON
NO
C
TEST FOR PROGRAMMED INTERRUPT (SEE FIGURE 21)
A
(SEE FIGURE 22)
YES
TEST IF OTHER PISW BITS ON
NO
MASK ALL OUT-OF-CORE INTERRUPTS
READ IN MAINLINE
UNMASK ALL OUT-OF-CORE INTERRUPTS
C
TEST FOR PROGRAMMED INTERRUPT (SEE FIGURE 21)

Figure 23. Exit from MIC After an Interrupt Has Been Serviced

The unmask will effectively unmask back to the last CALL MASK level.

A check is now made for the presence of additional process interrupts (that is, are other PISW bits on?). If they are indicated, an exit path is taken via (A) in the normal course of servicing process interrupts (previously explained). If no further process interrupts are present, all out-of-core interrupt levels are masked and variable core restored to its proper status which existed prior to the interrupt. The system is then unmasked to the user's status and an exit made via (C) and (B) -- see Figures 21 and 23.

The Case of I/O Interrupts. When an I/O device interrupt occurs, a similar procedure to that discussed in 1), 2), and 3) is adopted. In 3), it was mentioned that in the case of an I/O interrupt, the I/O servicing routine will be entered through its entry point in the Interrupt Branch Table (IBT). Some of the important aspects of the I/O device routine are discussed elsewhere in this section. The last instruction in an I/O device interrupt subroutine is an indirect branch BSI I 90) back to MIC. Before an exit is made through the common exit point ((B) -- see Figures 21 and 23) for all categories of interrupts, a check is performed to determine the presence of a programmed interrupt within the two groups of possible programmed interrupts -- group 1 (levels 0-13) and group 2 (levels 14-23). Only the bit associated with a level is tested. If a programmed interrupt is present, a branch is made to (A) and processing proceeds as for process interrupts. The I/O device interrupt, otherwise, undertakes to exit from MIC through the common route (B).

At this point, the FORTRAN I/O buffers are restored to their former state. All interrupt levels are masked, Index Registers 1, 2, and 3, and the accumulator, and words 54 and 55 are restored and the system is unmasked to the user's level. Programmed interrupts are now turned on (they were previously turned off) and a return is made to the interrupted mainline program.

Masking, Servicing, and Recording of Interrupts

An interrupt may occur at any time, but it will not be recognized by MIC until the level on which it is assigned is unmasked and of a higher priority than the current level of machine operation. It is the 1800 hardware, not MIC, that determines which level the interrupt is on. Interrupt levels are user-specified at system generation time. The user may delay any interrupt from being recognized by masking the level on which that interrupt has been assigned. For example, it may be to his advantage to delay the servicing of an interrupt to minimize core exchanges such as when it is known that a program is short and the interrupt can wait. In another situation, he may desire to prevent interrupts entirely from occurring, such as when a routine cannot be reentrant and may be called from more than one level. Once an interrupt has been recognized, MIC will determine if it is to be (1) serviced immediately or (2) recorded for servicing at a later time. Servicing an interrupt may be delayed by the user by simply setting a record option on that interrupt. The options of recording or servicing interrupts immediately may be changed from one mainline core load to another. This designation is made when the core load is initially built. MIC also services interrupts (a maximum of 384) in an optimized sequence within the user's specifications.

Masking of Interrupts

Interrupts can be prevented from occurring by masking. This is accomplished by using four real-time subroutines provided in TSX:

- CALL MASK
- CALL UNMK
- CALL SAVMK
- CALL RESMK

Call Mask. CALL MASK can be used to lock out for some time period those designated interrupt levels on which the user does not want interrupts to occur during some time-dependent programs. This routine gives him the facility to inhibit or mask out groups of interrupt levels (0-13; 14-23) or selectively chosen interrupt levels. The status of levels not designated remain unchanged. The format of this statement is:

CALL MASK (I, J)

Where I and J are integer expressions which designate the level(s) to be masked. Bits 0-13 of I refer to levels 0-13. Bits 0-9 of J refer to levels 14-23. Each one bit specifies a level to be masked. Both parameters are always required.

EXAMPLE 1. In this and following examples, DATA statements are used in conjunction with the CALL MASK and CALL UNMK statements to set up designated levels. See IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715.

The problem is to mask levels 5, 7, 11, 12, 21, 22 and 23.

DATA I, J/Z0518, Z01C0/

CALL MASK (I, J)

Call Unmask. CALL UNMK gives the user the ability to unlock an interrupt level -- that is, it allows interrupts to be recognized on a level. Thus, he may, if he wishes, selectively allow or unmask interrupts, one level at a time. This is a required routine (and procedure) for the initial core load -- the first core load called into the system by the Cold Start program. The statement format is

CALL UNMK (I, J)

Where I and J are integer expressions which designate the levels to be unmasked within the two groups of levels as for CALL MASK.

EXAMPLE 2. The problem is to unmask levels 1, 2, 3, 5, 12, and 21.

DATA I, J/Z7408, Z0100/

CALL UNMK (I, J)

From Examples 1 and 2 we see that

- Levels 1, 2, 3, 5, 12, and 21 are unmasked,
- Levels 7, 11, 22, and 23 are masked.
- Levels 4, 6, 8, 9, 10, 13-20 are unchanged.

The mask and unmask subroutines maintain a current record of the interrupt level mask status. This is necessary since the system sometimes masks all levels and then restores the status of these levels according to this record. The user should always mask and unmask via these routines to keep this record current.

EXAMPLE 3. The problem is to unmask all levels (as at cold start time).

CALL UNMK (-1, -1)

Call Save Mask. CALL SAVMK allows the user to save the masked condition (that is, the contents of the current mask words) that existed prior to his calling for masking. The statement format is:

CALL SAVMK (I, J)

Where I and J are integer variables that will receive the contents of the retained mask words.

For example, a mainline has just masked certain levels of interrupts. The user may not be aware of this condition -- that is, he may not know which bits are on (masked). So, he executes a CALL SAVMK to save this condition prior to masking those levels of interrupt he plans to have masked. When he is ultimately ready to unmask these levels, he executes a CALL RESMK which restores or returns the masked register to its original condition. This acts, effectively, as a mask and unmask routine and is closely analogous to the saving and restoring of registers, etc., during the handling of an interrupt.

Call Restore Mask. CALL RESMK is used to perform a mask and unmask operation to restore the interrupt mask register to its previously saved condition. The variables used as parameters are normally those named in a previous CALL SAVMK statement. Its format is:

CALL RESMK (I, J)

Where I and J are as for CALL MASK, except that each one bit specifies a level to be masked; each zero bit specifies a level to be unmasked.

EXAMPLE 4. The problem is to mask levels 5, 7, 9, 10, and 12; unmask all other levels.

DATA I, J/Z0568, Z0/

CALL RESMK (I, J)

Restrictions. It is not possible to unmask an out-of-core interrupt level:

1. while an out-of-core interrupt level specified on the System Director equate cards ICLL1-2 is being serviced,
2. while a mainline core load is being loaded by the Program Sequence Control (PSC) program -- e.g., by CALL CHAIN, CALL BACK, CALL SPECL.

Servicing of Interrupts

In the servicing of interrupts, the answers to three vital questions must be known:

1. What caused the interrupt?
2. How fast is its response?
3. How often does it occur?

In practice, the service action taken depends to a large extent on the frequency of occurrence of an interrupt, and the time required to service it -- that is, its servicing time span. There are, in general, four approaches in servicing interrupts:

- The servicing routine may reside in the skeleton.
- It may be located on disk as an interrupt core load.
- The user has the option to include the servicing routine as an integral part of a mainline core load.
- The user has the option to record the interrupt. That is, he may delay its servicing until it is cleared by a CALL CLEAR or serviced by a CALL QIFON.

CALL CLEAR -- Clear Recorded Interrupts

The CALL CLEAR Statement is used to ignore or clear interrupts which have occurred but which were recorded for later servicing. The statement format is:

CALL CLEAR (M, L, I, L, I, ...)

Where M = an integer constant which specifies the number of parameters to follow. If M = 0, all indicators specifying the recorded status are changed to indicate "not recorded".

and I = as for CALL QIFON (see Program Scheduling).

CALL CLEAR can be used in any process program.

The above four general approaches provide a variety of ways of handling a specific interrupt. For example, an INSKEL interrupt routine may set up a programmed interrupt for a level which is serviced by an out-of-core interrupt core load. This core load may, in turn, be made to queue a mainline core load or a series of mainline core loads to alter, say, the entire user control strategy.

Consider another example. A mainline core load may begin a chain of operations by setting up a programmed interrupt for a specific level. This interrupt may be recorded, or it may be immediately serviced.

The user will always obtain rapid and immediate servicing of interrupts if he (1) includes his interrupts as part of the System Skeleton, (2) does not record these interrupts. Interrupts that reside in core with the skeleton never require an exchange, while those that are included with a mainline core load may require an exchange if a nonprocess program is in memory on a time-sharing operation. If, however, time-sharing is not being used (that is, the mainline core load is in memory) or another interrupt serviced by an interrupt core load is in progress, interrupts in core with the mainline core load will be serviced almost immediately.

In general, therefore, interrupt servicing routines should be short in execution time. The reason for this is that the 1800 hardware locks out lower priority level interrupts for whatever time that is involved on that level. That portion of the interrupt routine that is not required for execution at this priority level should, therefore, be carried out either at the mainline level or at a lower priority level.

If mainline core loads are used to service interrupts through the queueing technique, then the user must ensure that his mainline core loads do not remain in execution for a period of time that is unacceptable to him prior to checking the Queue Table. A mainline core load may be interrupted by a CALL SPECL in such a core load (see Program Scheduling).

Recording of Interrupts

In general, interrupts may be recorded, that is, deferred service, under any of three different sets of circumstances:

1. When the user has one or more mainline core loads that must be executed within a certain time span.
2. When the user is adjusting or optimizing the process control and creating conditions which would cause interrupts to occur, and he elects to ignore them.

3. The user may wish to record interrupts for later servicing, but he prefers to do this through a CALL QIFON procedure rather than have them serviced on an interrupt level.

Interrupts to be recorded are entered on a *RCORD control card (in any order) and assembled at core load build time. The data set up in the card is later placed into the Interrupt Core Load Table from the Interrupt Status table (within each core load) by PSC.

The action of MIC when an interrupt occurs and the procedural flow through its servicing has already been described elsewhere in this section.

Rules Governing the Servicing of Interrupts

1. If an interrupt is serviced by a subroutine located in the variable area, it must be at a lower priority level (higher number) than the I/O device. This applies to:

   Interrupt and combination core loads
   Interrupt subroutines included with the mainline

The exception to this rule is that an interrupt must be on a level of priority lower than the I/O device it intends to use except for the disk and the 1053 typewriter. DISKN and TYPEN are so written that if either the disk or the typewriter detects that its call was executed from a level with a higher priority, it will remain in itself until the servicing operation is completed. This is achieved by sensing the appropriate Device Status Word (DSW).

2. If a servicing routine does not use any I/O device, it may be on any level, but the routine must be in the skeleton -- not in the variable area of core.
3. Interrupts on levels that are serviced by out-of-core interrupt core loads are serviced in the masked mode so that they cannot be interrupted by another interrupt serviced by an out-of-core routine. Only one level of exchange is maintained.

USE OF INTERVAL TIMERS

In most industrial control installations, some portion of the control of the user's system will require response in time -- that is, the user may want to schedule his programs periodically or at a specific time of day. For example, he may wish to print a shift log on a synchronous basis, say at 8 a.m., 4:30 p.m., and midnight each day; or he may take periodic scans of his process instrumentation once every five minutes; or there may be certain loops to time out.

An interval timer is, by definition, a clocking device which cycles a value contained in a full word of main storage. It thus provides a computer system with the ability to read elapsed time in second or millisecond increments, and to inform the system when a specified period of time has passed.

A simple cyclic timer serves, in effect, both as a basic interval counter and clock. In order to measure an elapsed time interval, a predetermined total count is loaded into the counter word storage by program control and a count down to zero is initiated. As the particular counter reaches zero, an internal interrupt signal is sent to the system.

Information about elapsed time and local time is often required by control computer systems to initiate hourly logs, to time the period between control actions on the process, for process data updating, etc. The time of day is required for printing logs, alarm records, and so on.

Clock interrupts can be used to start a scheduled computer operation. For example, in the control of a complex distillation plant process, periodic interrupts have been used to initiate the recalculation of the reflux ratio required to maintain a desired separation in the tower. In this situation, control of a dependent process quantity is possible through a periodic reexamination of process conditions requiring extensive computer time.

To accomplish the above, the Interval Timer Control (ITC) program provides for FORTRAN language control of three hardware interval timers, A, B, and C which operate on various user-specified time bases (see Table 1). Timers A and B are available to the user, while Timer C is used exclusively by TSX for time-sharing control purposes and as a real-time clock. Furthermore, Timer C is expanded into nine additional programmed interval timers -- thus making available to the user a total of 11 interval timers. As shown in Figure 24, each interval timer is assigned a fixed location in core storage.

ITC also performs three additional functions:

- Resets the Operations Monitor during time-sharing
- Tests for no response from 1053 printers
- Performs end of time-sharing

Name	Core Storage Location
Machine Timers	
A	00004
B	00005
C	00006
Programmed Timers	
1	00062
2	00065
3	00068
4	00071
5	00074
6	00077
7	00080
8	00083
9	00086
Time-Sharing Clock	00089

Figure 24. Timer Locations in Core Storage

The establishment of the two principal time bases, the Primary (or Interrupt) Time Base and the Secondary (or Programmed) Time Base, and their relationships to the system are discussed in the section, System Design Considerations: System Director.

Each timer is assigned to a wired-in time base by the user at system generation time, selectable from the table of available time bases given in Table 1.

The .125ms time base is available only on a 2usec machine; the 128ms time base, only on a 4usec machine. Each timer is assigned a permanent time base by the user. Note that a different time base can be selected for each timer, but all three timers (A, B, and C) must be assigned to the same interrupt level. In order to schedule programs based on hours, minutes, or seconds, the wired-in time base for interval timer C must be an even divisor of one second (e.g., .5, 1, 2, 4, 8). The servicing of all interrupts is controlled by ITC.

Hardware Timers A and B

CALL TIMER

In order to use timers A and B, the system provides a basic call statement:

CALL TIMER (NAME, I, INT)

where

NAME	= Name of the user's subprogram that is executed when the specified time elapses. Note that NAME must also appear in a FORTRAN EXTERNAL statement (see IBM 1130/1800 Basic FORTRAN IV Language, Form No. C26-3715).
I	= An integer expression whose value must be: 1 for Timer A (word 00004) 2 for Timer B (word 00005)
INT	= A user-assigned positive integer expression which specifies the number of interval counts before the user's subprogram is executed.

The subprogram specified in a CALL TIMER statement must be in core storage when the interrupt generated by the timer is recognized. The interrupt occurs when the time specified has elapsed, but it is only recognized

1. When the level of current operation is lower than the timer interrupt level, and
2. If the timer level is unmasked.

Table 1. Table of Available Timer Time Bases

Core Storage Cycle Times	Available Time Bases (In Milliseconds)									
2 μsec	.125	.25	.5	1	2	4	8	16	32	64
4 μsec	.25	.5	1	2	4	8	16	32	64	128

At the end of the elapsed time, the timer resets itself. Note that, when zero time has been reached, the timer continues to operate -- that is, zero is not a not-busy condition.

In the section System Design Considerations: System Director, it is pointed out that it is the user's responsibility to ensure that the mainline program which requested the timer statement remain in core until the end of the elapsed specified time -- that is, until the timer times out. He achieves this either by

1. Including the subprogram in the Skeleton, or by
2. Masking out all out-of-core interrupt levels, and forbidding a core load exit until the timer interrupts.

Unless previously loaded with the System Skeleton, the subprogram is automatically loaded with the calling mainline core load.

In addition, periodic programs (that is, programs initiated by interval timers) should not, as a rule, be executed on the timer level: they should make use of the programmed interrupt technique.

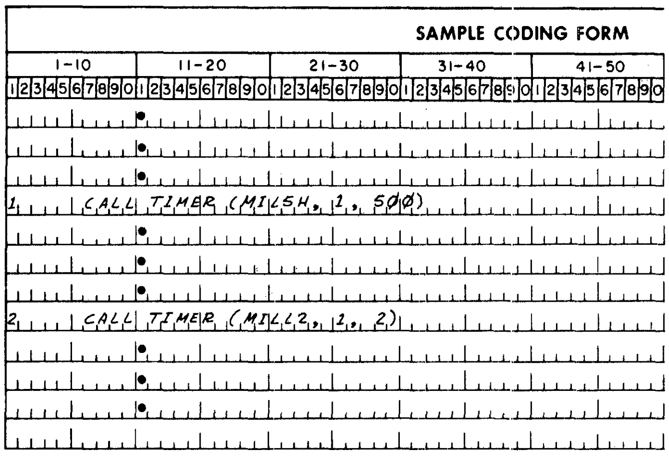
The following examples assume that the timers specified are called from only one level. If possible, it is preferable not to share timers among two different programs.

EXAMPLE 1. Assume hardware Timer A is wired for the .125ms time base.

CALL TIMER (SCAN1, 1, 35)

When this statement is executed, ITC initializes Timer A (by setting it to -125) and returns control to the next executable instruction following the CALL TIMER statement. When the Primary (or Interrupt) Time Base (= 35 X .125 = 4.375ms) elapses, an interrupt occurs and control passes to the subprogram named SCAN1.

EXAMPLE 2. Assume hardware Timer A is wired for the 1ms time base.



SAMPLE CODING FORM

```
1     CALL TIMER (MIL5H, 1, 5ØØ)
2     CALL TIMER (MILL2, 1, 2)
```

If we assume that the Primary Time Base (= 500ms) for statement 1 in the above coding has not elapsed, the Timer A interrupt will occur 2ms after execution of statement 2 when subprogram MILL2 will be executed. Subprogram MIL5H will never be executed because Timer A was reset before the 500ms time elapsed. Although this condition can be prevented (see Example 3), its logic can prove useful under certain practical conditions.

EXAMPLE 3. Assume identical conditions as for Example 2. This example illustrates the use of the LD functional subroutine in testing for a timer-busy condition.

The format of this function is:

LD(I)

where

I	= A user-assigned integer expression that specifies a core storage address. The contents of this address are moved to the accumulator. This permits a test for busy, etc., of known locations outside of the program area. Timer storage locations are given in Figure 24.

SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
12345678901234567890123456789012345678901234567890
          .
          .
          .
1     CALL TIMER (MIL5H, 1, 5ØØ)
          .
          .
          .
          .
          .
2     IF(LD(4))2,3,3
3     CALL TIMER (MILL2, 1, 2)
```

Statement 2 tests if Timer A is busy. If it is busy (that is, negative in core location 00004), a programmed loop is activated until Timer A is no longer busy (that is, when subprogram MIL5H is executed) at which time statement 3 is processed.

EXAMPLE 4. Another example is given to illustrate the use of the LD subroutine function for a test for timer-busy condition.

This test is required if subprogram SUBR7 is not in the skeleton and time-sharing is utilized.

In this example, statement 12 tests if Timer A is busy, and waits until subprogram SUBR7 has been executed before passing to the CALL VIAQ statement.

SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
12345678901234567890123456789012345678901234567890
          .
          .
          .
      CALL SAVMK (I,J)
      CALL MASK (K,L)
      CALL TIMER (SUBR7, 1, 1Ø)
          .
          .
          .
12    IF(LD(4))12, 13, 13
13    CALL RESMK (I,J)
      CALL VIAQ
```

NOTE: The execution of a machine interval timer busy-test using the LD (I) functional subroutine in an IF statement may fail to indicate the correct busy status if (1) the timer interrupt occurs immediately after the loading of the timer not-busy indication (a zero), and (2), in servicing the interrupt, the timer is reinitialized on another level.

Thus, when a timer is shared by different levels, a solution (see below) would be to follow the first busy-test by a second busy-test in order to prevent an interrupt out of the busy-test.

SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
12345678901234567890123456789012345678901234567890
9     CALL UNMK(I,J)
         .
         .
         .
12    IF(LD(5))12,13,13
13    CALL MASK (I,J)
      IF(LD(5))9,14,14
14    CALL TIMER ( , , )
      CALL UNMK(I,J)
```

Notice that although the not-busy status remains in the accumulator after the return from the interrupt, it will be initialized for testing in the following load instruction.

Real-Time Clock

ITC also provides a programmed real-time clock which keeps time on a 24-hour basis and is updated each time Timer C decrements to zero (that is, it is incremented from 00.000 to 23.999; then returns to 00.000). The clock accuracy is a function of the Primary (or Interrupt) Time Base discussed in the section System Design Considerations: System Director

CALL SETCL -- Set-up Programmed Real-time Clock

Note that the clock is set at cold start time (a user option), but if it is required to be set at any other time through a user program, the following statement is provided.

CALL SETCL (I)

where

I	=	A user-assigned integer expression specifying the time of day setting desired in hours and thousandths of hours (e.g., 8 a.m. = 08000; 10.45 a.m. = 10750)

CALL CLOCK -- Read Programmed Real-time Clock

If the user desires to read the clock, say, for time-recording of his output to the printer, disk, etc., he does so through a

CALL CLOCK (I)

where

I	=	A user-assigned integer variable which indicates the core location where the readout time is stored.

Note that the clock is also used by the Error Alert Control (EAC) Program to time-stamp error messages.

Programmed Timers

The mechanism of programmed timers is covered in the section System Design Considerations: System Director.

CALL COUNT

Programmed interval timers are controlled by the following statement.

CALL COUNT (IN, I, INB)

where

IN	=	A user-assigned integer constant or integer variable that specifies the number (in the range 0-31) of the program to be executed or recorded when the specified time elapses. The number is assigned at System Skeleton build time. Program numbers are used instead of names to provide the record interrupt option.
I	=	An integer expression, identifying the number (1-9) of the programmed timer.
INB	=	A user-assigned expression that specifies the number of interval counts before the called program is executed. This number is a function of the Secondary (or Programmed) Time Base.

An additional programmed timer is used as the time-sharing control timer for the allocation of time slicing for non-process operations (see Use of Time Sharing).

EXAMPLE 5. The problem is to queue an analog scan program every five minutes with a priority of 7 if JTEST (a programmed indicator in INSKEL COMMON) is set to zero; if it is non-zero, queue the same program every minute with a priority of 1.

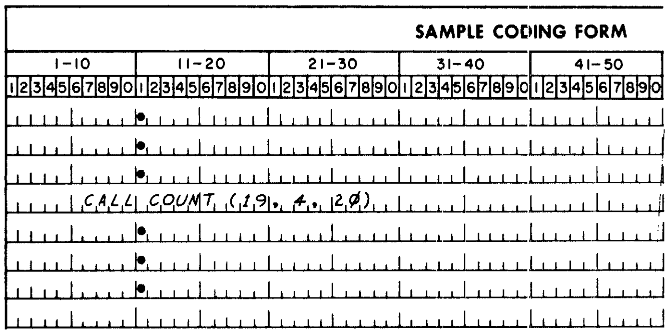
Assume the following:

1. Subroutine 19 is SUBROUTINE A which was included in the Skeleton at Skeleton build time by an include card

   *INCLD A/2703

   thus assigning it as count routine number 19.
2. Primary Time Base = 8ms (Timer C wired time base) X 125 (user-assigned number) = 1 second
   Secondary Time Base = 1 (Primary Time Base) X 15 (user-assigned number) = 15 seconds

To solve the problem, a CALL COUNT statement must be given in a mainline core load, thus:


SAMPLE CODING FORM

```
     CALL COUNT (19, 4, 2Ø)
```

This designates that subroutine 19 is to be called in 5 minutes; thereafter, the subroutine calls itself within the specified time period. Its coding is shown in Figure 25.

SCAN is the name of a mainline core load that will be executed at mainline level as the result of a CALL VIAQ when SCAN is the highest priority entry in the queue.

In order to effect immediate execution of the scan routine, the CALL QUEUE statement may be replaced by a CALL LEVEL statement to cause an interrupt on a lower level. This allows the user the flexibility of executing the SCAN routine either as an interrupt core load, an INSKEL interrupt servicing routine, or as a routine included with a mainline. The advantage is that the timer interrupt level is not tied up. It also gives the user the ability to call other I/O devices within the SCAN routine.

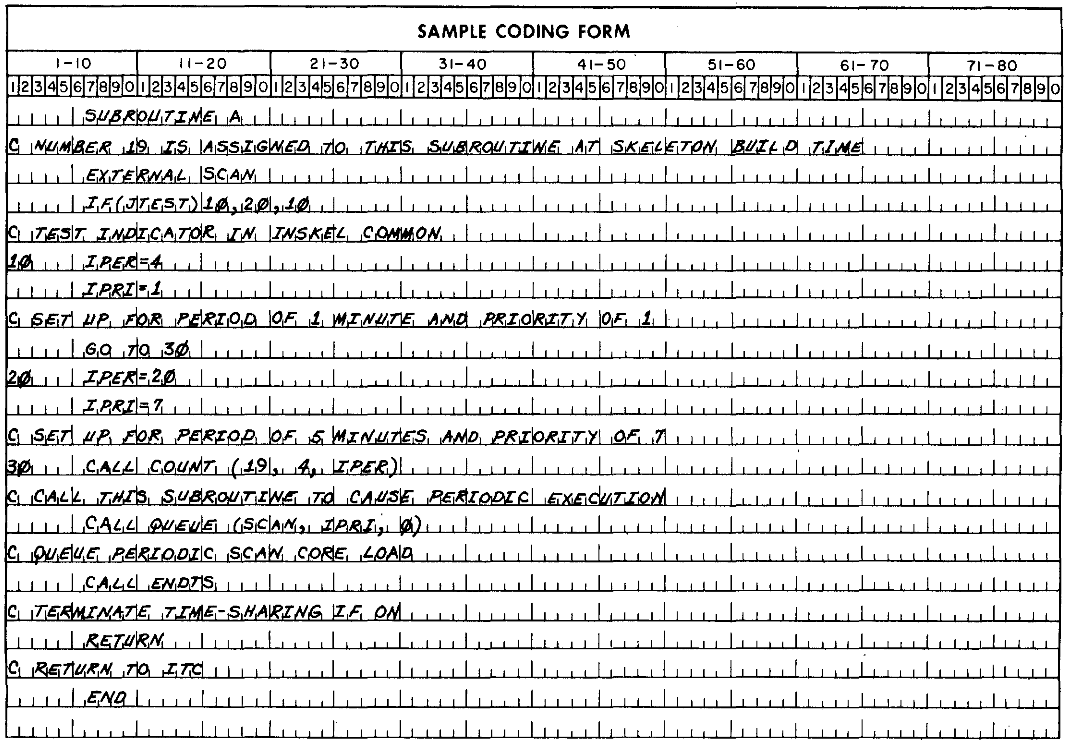
If the time-sharing mode is not used, the CALL ENDTS statement has no effect. If it is used, the time-sharing clock is set to zero and a return made to the calling program. See Use of Time-Sharing for further action.

A further example is given elsewhere in this section (see Program Scheduling).

Table 2 provides a ready comparison of the salient features in the usage of interval timers and programmed timers.

USE OF TIME-SHARING

In many industrial control installations, the user will have a large amount of time that is not utilized by the process being controlled. To allow him to make effective use of this time, the time-sharing feature of the TSX system gives him the ability to compile, assemble, and simulate without taking the system off-line. In this manner, low-priority jobs are automatically interrupted whenever the need arises to execute a higher-priority task. In addition, the inclusion of this feature gives the user the capability of modifying the logic of his control strategy.


SAMPLE CODING FORM

```
      SUBROUTINE A
C NUMBER 19 IS ASSIGNED TO THIS SUBROUTINE AT SKELETON BUILD TIME
      EXTERNAL SCAN
      IF(JTEST)1Ø,2Ø,1Ø
C TEST INDICATOR IN INSKEL COMMON
1Ø    IPER=4
      IPRI=1
C SET UP FOR PERIOD OF 1 MINUTE AND PRIORITY OF 1
      GO TO 3Ø
2Ø    IPER=2Ø
      IPRI=7
C SET UP FOR PERIOD OF 5 MINUTES AND PRIORITY OF 7
3Ø    CALL COUNT (19, 4, IPER)
C CALL THIS SUBROUTINE TO CAUSE PERIODIC EXECUTION
      CALL QUEUE (SCAN, IPRI, Ø)
C QUEUE PERIODIC SCAN CORE LOAD
      CALL ENDTS
C TERMINATE TIME-SHARING IF ON
      RETURN
C RETURN TO ITC
      END
```

Figure 25. Subroutine A for Example 5 -- Queueing an Analog Scan Program

Table 2. Comparison of Timers

INTERVAL TIMERS 1 & 2	USED WHEN SHORTER TIME BASE IS SPECIFIED	SUBROUTINE CALLED MUST BE IN CORE (IN SKELETON OR INCLUDED WITH MAINLINE) WHEN TIMER ELAPSES	CANNOT BE A RECORDED INTERRUPT	SUBROUTINE IS EXECUTED ON INTERRUPT LEVEL OF INTERVAL TIMERS	EXIT WITH A RETURN STATEMENT
PROGRAMMED TIMERS	USED WHEN LONGER TIME BASE IS NEEDED (E.G., HOURS)	SUBROUTINE MAY OR MAY NOT BE IN CORE WHEN TIMER ELAPSES	IF SUBROUTINE IS NOT IN CORE, IT IS HANDLED AS A RECORDED INTERRUPT	MAY BE EXECUTED ON INTERRUPT LEVEL OR MAINLINE LEVEL	EXIT WITH A RETURN OR CALL VIAQ

Methods of Initiating Time-Sharing

Time-sharing can be initiated in two ways: selectable method (CALL SHARE) and automatic method (CALL VIAQ).

Selectable Method -- CALL SHARE

The user will know at some predetermined point in his program that he wishes to discontinue being in the process mode for a specific period of time. He therefore enters the time-sharing mode by the execution of a CALL SHARE (that is, he gives up control to the Nonprocess Monitor via the CALL SHARE). This statement may be part of the user's process program intended for those special applications where time-sharing is desired without the use of the queueing technique. Its format is as follows:

CALL SHARE (I)

Where I is an integer expression which specifies the number of time intervals allowed for the nonprocess program operation. The basic time interval is assigned by the user at system generation time (see System Design Considerations - System Director; also Use of Interval Timers).

The meaning of the I parameter is clarified by the following example.

EXAMPLE 1. Assume that the secondary time base is 15 seconds (see Use of Interval Timers). Then

Time-Sharing Interval Requested	Required Statement
1 minute	CALL SHARE (4)
5 minutes	CALL SHARE (20)
30 seconds	CALL SHARE (2)
1.75 minutes	CALL SHARE (7)

The time-shared operation is terminated whenever the time interval specified by the user has elapsed; it is usually not terminated before. Thus, if 1 minute of time-sharing is indicated, it is usually 1 minute before control is returned to the next executable instruction following the CALL SHARE statement. The exchange time is not part of the 1 minute specification. This 1 minute is the length of the time in the share mode. All interrupt time is alloted against this 1 minute span.

Note that the Nonprocess Monitor will perform a WAIT operation if there are no off-line jobs for execution. Also, interrupts will be serviced as they occur. If an interrupt routine recognizes a need for the process program to resume operation, it can terminate the time-sharing mode by executing the following call:

CALL ENDTS

CALL ENDTS can be used only in an interrupt routine where it sets the time-sharing clock to indicate zero time. The first Timer C interrupt that occurs after control is returned to the nonprocess program causes the time-sharing operation to be terminated; control then reverts to the process mainline program. Note also that whenever time-sharing is not in force the CALL ENDTS statement is ineffective.

Automatic Method -- CALL VIAQ

The second method uses the queueing technique to load a mainline or combination core load when the Core Load Queue Table is empty, by executing a CALL VIAQ (See Program Scheduling).

Note that a CALL VIAQ (when referenced) forces a CALL SHARE statement for execution when the queue is empty only if the user has indicated through the use of the Console Interrupt button, with sense switch 7 on, that batch work is to be carried out. As a result, the process core load which is in progress, or which has just been completed, is saved on disk and control transferred to the Non-process Monitor (or the nonprocess core load if one had been interrupted and stored on disk). The period of time allocated to time-sharing is specified by the user in a System Director equate card, TISHA, at system generation time. The computer remains in the nonprocess mode for this specified period unless a CALL ENDTS is executed by an interrupt routine.

At the completion of the specified time, another CALL VIAQ is automatically forced by the system. If, in the meantime, a core load has been queued, it is then executed. If the queue remains unchanged (that is, nothing has been added to it), another time-sharing operation will be triggered.

If, at the end of a nonprocess job, the // END OF ALL JOBS card indicates that there is no further nonprocess work for execution, the VIAQ routine will WAIT until either some addition has been made to the queue or the Console Interrupt (C.I.) button is again depressed for the commencement of a new nonprocess job.

This method of entering time-sharing is, in practice, preferred to CALL SHARE. CALL SHARE may, however, be desirable in certain special situations.

Two additional functions performed by the Time-Sharing Control (TSC) program are CALL LINK and CALL EXIT when these are referenced from nonprocess programs.

EXAMPLE 2. (See Program Listing No. 1). In order to illustrate some of the many TSX usages without complex FORTRAN/Assembler language coding, the following example was devised. Note that in this example, the system and list printers have been defined as the same device (1443). In actual practice, the system printer would be a 1053; the list printer, a 1443 or another 1053.

Three analog inputs, A, B, and C, are to be read at 15-second intervals. After C has been read, linear interpolation is used between point A and point B, and between point B and point C. The values A, B, and C are temperatures: the temperatures between A and B, and B and C are linear. The point at which temperature A is taken is 25 feet away from the point where temperature B is taken; similarly for B and C.

A temperature histogram showing temperature versus distance is to be printed on the list printer.

A nonprocess program is to be written which simply lists numbers: this program is to be executed in the time-sharing mode.

Timer 2 is used to produce an interrupt every 15 seconds so that one of the three analog inputs may be read.

The skeleton contains a timer service subroutine for Timer 2, called SCAN, which calls programmed interrupt level 7 when 15 seconds have elapsed (that is, SCAN executes a CALL LEVEL (7)). Timer 2 has a base (TBASE) of 1 millisecond.

The problem was solved under TSX using the in-skeleton subroutine SCAN and the following five core loads:

COLDC
WAITC
READC
CALCC
SHOWC

Figure 26 illustrates the general problem logic flow.

COLDC (referred to at execution time as C/L #1). This is a mainline core load which is directly called by the cold start program. Its primary function is to unmask all interrupt levels, set timer to 15 seconds, and chain to core load WAITC.

WAITC (referred to at execution time as C/L #2). This core load merely calls VIAQ which results in either a queued program being executed, or the beginning of time-sharing.

READC (referred to at execution time as C/L #3). This is the solitary interrupt core load which is executed on level 7. The SCAN routine in skeleton executes a programmed interrupt to level 7 each time the 15-second interval elapses. The *STORECI control card for this core load contains level and bit indicators equal to 2407 -- which indicates programmed interrupt level 7.

When this core load is executed, an indicator named ICNT, which is in INSKEL COMMON, is interrogated. If this indicator is 1, the first point A is read, timer 2 is reset (for another 15-second interval), and the core load exits by way of a CALL INTEX.

If the indicator is 2, the second point B is read, the timer is reset, and the core load exits.

If the indicator is 3, the third point C is read, the timer is reset, two core loads CALCC and SHOWC are queued, time-sharing is terminated, and the core load exits via a CALL INTEX.

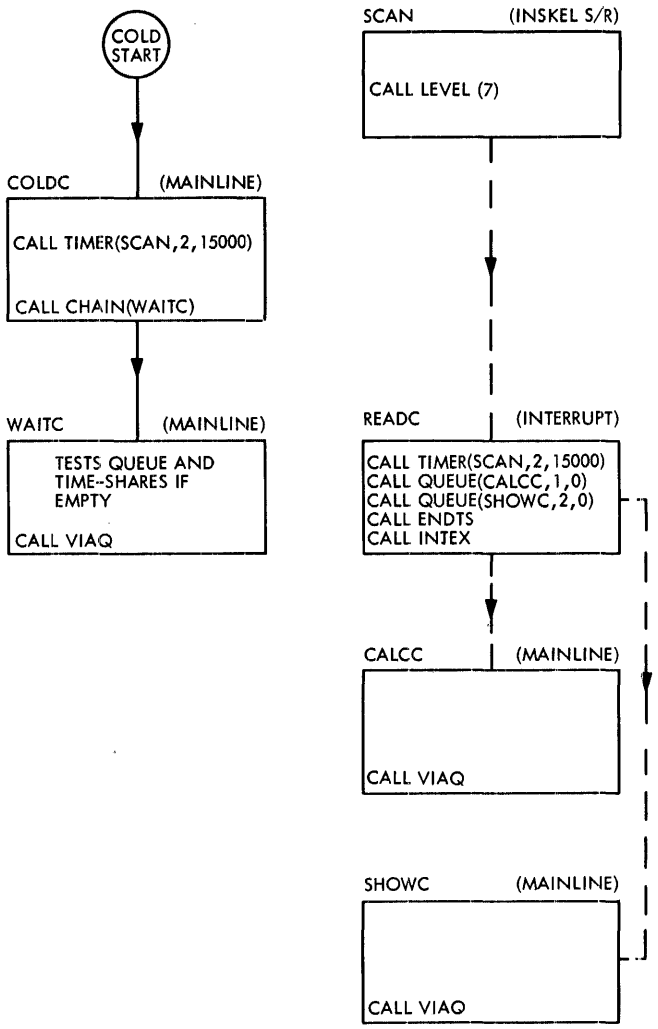


Figure 26. General Problem Logic Flow -- Example 2

CALCC (referred to at execution time at C/L #4). CALCC takes the three analog readings, A, B, and C, which have been stored in INSKEL COMMON, interpolates and stores the 51 results back into INSKEL COMMON.

The core load is executed by a CALL VIAQ.

SHOWC (referred to at execution time as C/L #5). SHOWC takes the 51 interpolated results from INSKEL COMMON and outputs a scaled histogram on the list printer. It then calls VIAQ.

NOTE: Each core load prints a message on entry to and on exit from the core load itself. This message identifies the core load as C/L 1, C/L 2, C/L 3, C/L 4, or C/L 5.

This diagnostic message is accomplished by a CALL-type FORTRAN subroutine which is included in the skeleton. Its format is as follows:

CALL ENT (I, J)

where ENT is the name of this subroutine. Either of two messages, depending on the parameters I and J, will be printed:

A) ENTERED C/L NO. . . . .
B) EXITED C/L NO. . . . .

ENTERED will be printed when I = 1.
EXITED will be printed when I = 2.

J is the core load identification number as follows:

J = 1 = COLDC

J = 2 = WAITC

J = 3 = READC

J = 4 = CALCC

J = 5 = SHOWC

The on-line results on the list printer (Program Listing No. 1) also clearly indicate when time-sharing has taken place.

PROGRAM LISTING NO. 1: EXAMPLE 2

```
        FLET

PACK LABEL                                                                   (Note: This is the state of FLET before compilations
 00000                                                                               begin )

.FIOS 001B  03A0     /EPDM 7FFF  03BB     /EPSV 0780  0422     /INSV 2280  0428     /NPSV 4000  0444     .MESS 0010  0478
DUMMY 0092  0488     DUMIN 005A  0489     NONPR 00F0  048A     NP    0098  048B     9DUMY 00EC  048C     /SPSV 4000  0578
/PRSV 4000  05AC     .SKEL 0038  05E0     .EPRG 0022  0618     /CLST 0780  063A     .E    00F0  0488

DUP FUNCTION COMPLETED




// JOB
// FOR COLDP
*IOCS(1443PRINTER)
*LIST ALL



      EXTERNAL SCAN,WAITC
      COMMON/INSKEL/I1,I2,I3,INCNT
      CALL UNMK(-1,-1)
      CALL ENT(1,1)
      INCNT=1
      CALL TIMER (SCAN,2,15000)
      CALL ENT(2,1)
      CALL CHAIN (WAITC)
      END


VARIABLE ALLOCATIONS
 I1    =FFFF* I2    =FFFE* I3    =FFFD* INCNT=FFFC*

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 SCAN     WAITC    UNMK     ENT      TIMER    CHAIN    PRNTN    EBPRT

INTEGER CONSTANTS
     1=0004        2=0005   15000=0006

CORE REQUIREMENTS FOR COLDP
 COMMON       0  INSKEL COMMON       4  VARIABLES       4  PROGRAM      40

 END OF COMPILATION


COLDP
DUP FUNCTION COMPLETED
// DUP
*STORECIM M          COLDC COLDP COLDC
*CCEND

 CLB, BUILD COLDC

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 4002  000C
 *IBT TABLE 400E  001D
 *FIO TABLE 402B  0010
 *ETV TABLE 403B  000F
 *VTV TABLE 404A  001E
 *IST TABLE 4068  0036
 *PNT TABLE 409E  000C
 MAIN COLDP 40B1
 PNT  COLDC 40A0
 PNT  COLDC 40A4
 CALL UNMK  40D6
 CALL ENT   413D
 CALL TIMER 415C
 PNT  WAITC 40A8
 LIBF SUBIN 41B2  404A
 LIBF COMGO 41EC  404D
 LIBF MWRT  43C8  4050
 LIBF MIOI  447E  4053
 LIBF MCOMP 4455  4056
 LIBF IOU   487A  4059
 CALL IOFIX 4932
 CALL BT1BT 4962
 CALL SAVE  48CE
 LIBF ADRCK 49C6  405C
 LIBF FLOAT 4A18  405F
 LIBF IFIX  4A34  4062
 LIBF NORM  4A60  4065
 CORE       4A8E  3572

 CLB, COLDC LD XQ

D 45 CORELOADS NOT FOUND
WAITC
DUP FUNCTION COMPLETED
```

This is a genuine TSX warning message. It indicates that core load WAITC was not built at this stage.

```
// JOB
// FOR WAITP
*LIST ALL
*IOCS (1443PRINTER)



      CALL ENT(1,2)
      CALL ENT(2,2)
      CALL VIAQ
      END


FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 ENT      VIAQ     PRNTN    EBPRT

INTEGER CONSTANTS
     1=0000       2=0001

CORE REQUIREMENTS FOR WAITP
 COMMON       0  INSKEL COMMON       0  VARIABLES       0  PROGRAM      12


 END OF COMPILATION


WAITP
DUP FUNCTION COMPLETED
// DUP
*STORECIM M          WAITC WAITP COLDC
*CCEND

 CLB, BUILD WAITC

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 4002  000C
 *IBT TABLE 400E  001D
 *FIO TABLE 402B  0010
 *ETV TABLE 403B  000F
 *VTV TABLE 404A  001E
 *IST TABLE 4068  0036
 *PNT TABLE 409E  0008
 MAIN WAITP 40A8
 PNT  WAITC 40A0
 PNT  COLDC 40A4
 CALL ENT   40CF
 CALL VIAQ  40EE
 LIBF SUBIN 414E  404A
 LIBF COMGO 4188  404D
 LIBF MWRT  4364  4050
 LIBF MIOI  441A  4053
 LIBF MCOMP 43F1  4056
 LIBF IOU   4816  4059
 CALL IOFIX 48CE
 CALL BT1BT 48FE
 CALL SAVE  486A
 LIBF ADRCK 4962  405C
 LIBF FLOAT 49B4  405F
 LIBF IFIX  49D0  4062
 LIBF NORM  49FC  4065
 CORE       4A2A  35D6

 CLB, WAITC LD XQ

DUP FUNCTION COMPLETED
```

```
// JOB
// FOR READP
*IOCS(1443PRINTER)
*LIST ALL


      EXTERNAL SCAN,CALCC,SHOWC
      COMMON/INSKEL/IA1,IA2,IA3,ICNT
      CALL ENT(1,3)
      L=ICNT
      GO TO (5,10,15),L
    5 K=76
      GO TO 20
   10 K=79
      GO TO 20
   15 K=127
   20 CALL AIP(0,JTEST)
      GO TO (25,30),JTEST
   25 GO TO 20
   30 CALL AIP(01000,ITEMP,K)
   70 CALL AIP(0,JTEST)
      GO TO (71,72),JTEST
   71 GO TO 70
   72 GO TO (35,40,45),L
   35 IA1=ITEMP
      GO TO 50
   40 IA2=ITEMP
      GO TO 50
   45 IA3=ITEMP
   50 WRITE(3,100)  ICNT
  100 FORMAT ('  ICNT=',I3)
      ICNT=ICNT+1
      CALL TIMER (SCAN,2,15000)
      GO TO (55,55,55,60),ICNT
   55 CALL ENT(2,3)
      CALL INTEX
   60 ICNT=1
      CALL QUEUE(CALCC,1,0)
      CALL QUEUE(SHOWC,2,0)
      CALL ENDTS
      CALL ENT(2,3)
      CALL INTEX
      END

VARIABLE ALLOCATIONS
 IA1   =FFFF* IA2   =FFFE* IA3   =FFFD* ICNT =FFFC* L     =0000  K     =0001  JTEST=0002   ITEMP=0003

STATEMENT ALLOCATIONS
 100   =000D  5     =0023  10    =0029  15    =002F  20    =0033  25    =003D  30    =003F  70    =0044  71    =004E  72    =0050
 35    =0057  40    =005D  45    =0063  50    =0067  55    =0081  60    =0087

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 SCAN     CALCC    SHOWC    ENT      AIP      TIMER    INTEX    QUEUE    ENDTS    COMGO    MWRT     MCOMP    MIOI     PRNTN    EBPRT

INTEGER CONSTANTS
     1=0004        3=0005       76=0006       79=0007      127=0008        0=0009     1000=000A       2=000B  15000=000C

CORE REQUIREMENTS FOR READP
 COMMON       0  INSKEL COMMON       4  VARIABLES       4  PROGRAM     156


 END OF COMPILATION


READP
DUP FUNCTION COMPLETED
// DUP
*STORECIM I          READC READP         2407
*CCEND
```

READC is an interrupt core load responding to a programmed interrupt on level 07.

```
 CLB, BUILD READC


 ROC  ANINT 0023  LEV.0

 CORE LOAD  MAP
 TYPE NAME  ARG1   ARG2
 *CDW TABLE 4002  000C
 *IBT TABLE 400E  001D
 *FIO TABLE 402B  0010
 *ETV TABLE 403B  000F
 *VTV TABLE 404A  0021
 *PNT TABLE 406C  000C
 MAIN READP 408C
 PNT  READC 406E
 CALL ENT   4135
 LIBF COMGO 4154  404A
 CALL AIP   41A6
 LIBF MWRT  435C  404D
 LIBF MIOI  4412  4050
 LIBF MCOMP 43E9  4053
 CALL TIMER 480E
 CALL QUEUE 4864
 PNT  CALCC 4072
 PNT  SHOWC 4076
 CALL ENDTS 4926
 LIBF SUBIN 4930  4056
 CALL QZ010 496A
 CALL QZERQ 498E
 LIBF AIPTN 49CC  4059
 LIBF IOU   4A4E  405C
 CALL IOFIX 4B06
 CALL BT1BT 4B36
 CALL SAVE  4AA2
 LIBF ADRCK 4B9A  405F
 LIBF FLOAT 4BEC  4062
 LIBF IFIX  4C08  4065
 CALL GAGED 4C34
 CALL UNGAG 4C45
 CALL ANINT 4C54
 LIBF NORM  4D90  4068
 CORE       4DBE  1242

 CLB, READC LD XQ

D 45 CORELOADS NOT FOUND
CALCC SHOWC
DUP FUNCTION COMPLETED




// JOB
// FOR CALCP
*LIST ALL
*IOCS (1443PRINTER)



      DIMENSION N(51)
      COMMON/INSKEL/J1,J2,J3,ICNT,N
      CALL ENT(1,4)
      WRITE (3,6) J1,J2,J3
    6 FORMAT ('  READINGS',3I10)
      N(1)=J1
      N(26)=J2
      N(51)=J3
      DO 4 I=2,25
    4 N(I)=N(1)+((N(26)-N(1,,/25)*(I-1)
      DO 5 I=27,50
    5 N(I)=N(51)+((N(26)-N(51))/25)*(51-I)
      WRITE (3,7) (N(I),I=1,51)
    7 FORMAT (12I10)
      CALL ENT(2,4)
      CALL VIAQ
      END



VARIABLE ALLOCATIONS
 J1    =FFFF* J2    =FFFE* J3    =FFFD* ICNT =FFFC* N     =FFFB* I     =0002

STATEMENT ALLOCATIONS
 6     =000E  7     =0017  4     =003E  5     =006A

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 ENT      VIAQ     ISTOX    MWRT     MCOMP    MIOIX    MIOI     SUBSC    PRNTN    EBPRT
```

```
INTEGER CONSTANTS
     1=0006       4=0007       3=0008       2=0009      25=000A      27=000B      50=000C      51=000D

CORE REQUIREMENTS FOR CALCP
 COMMON        0  INSKEL COMMON      56  VARIABLES       6  PROGRAM     170


 END OF COMPILATION


CALCP
DUP FUNCTION COMPLETED
// DUP
*STORECIM M          CALCC CALCP COLDC
*CCEND

 CLB, BUILD CALCC

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 4002  000C
 *IBT TABLE 400E  001D
 *FIO TABLE 402B  0010
 *ETV TABLE 403B  000F
 *VTV TABLE 404A  0027
 *IST TABLE 4071  0036
 *PNT TABLE 40A8  0008
 MAIN CALCP 40CA
 PNT  CALCC 40AA
 PNT  COLDC 40AE
 CALL ENT   417D
 LIBF MWRT  4326  404A
 LIBF MIOI  43DC  404D
 LIBF MCOMP 43B3  4050
 LIBF ISTOX 47D8  4053
 LIBF SUBSC 47F8  4056
 LIBF MIOIX 43E8  4059
 CALL VIAQ  4824
 LIBF SUBIN 4884  405C
 LIBF COMGO 48BE  405F
 LIBF IOU   4910  4062
 CALL IOFIX 49C8
 CALL BT1BT 49F8
 CALL SAVE  4964
 LIBF ADRCK 4A5C  4065
 LIBF FLOAT 4AAE  4068
 LIBF IFIX  4ACA  406B
 LIBF NORM  4AF6  406E
 CORE       4B24  34DC

 CLB, CALCC LD XQ

DUP FUNCTION COMPLETED




// JOB
// FOR SHOWP
*IOCS (1443PRINTER)
*LIST ALL


      DIMENSION N(51),M(51),L(120)
      COMMON/INSKEL/I1,I2,I3,ICNT,N
      CALL ENT(1,5)
      DO 2 IK=1,120
    2 L(IK)=0
      DO 3 I=1,51
      MI=N(I)/300
    3 M(I)=IABS(MI)
      DO 4 J=1,51
      K=M(J)/2
    4 WRITE (3,100) J,(L(I),I=1,K)
  100 FORMAT (I3,1X,58I2)
      CALL ENT (2,5)
      CALL VIAQ
      END

VARIABLE ALLOCATIONS
 I1    =FFFF* I2    =FFFE* I3    =FFFD* ICNT =FFFC* N     =FFFB* M     =0032  L     =00AA  IK    =00AB  I     =00AC  MI    =00AD
 J     =00AE  K     =00AF
```

```
STATEMENT ALLOCATIONS
 100  =00BA  2    =00C7  3    =00E9  4    =0109

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 ENT      IABS     VIAQ     ISTOX    MWRT     MCOMP    MIOIX    MIOI     SUBSC    PRNTN    EBPRT


INTEGER CONSTANTS
     1=00B2       5=00B3    120=00B4       0=00B5      51=00B6    300=00B7       2=00B8       3=00B9

CORE REQUIREMENTS FOR SHOWP
 COMMON      0  INSKEL COMMON     56  VARIABLES    178  PROGRAM    128


 END OF COMPILATION


SHOWP
DUP FUNCTION COMPLETED
// DUP
*STORECIM M          SHOWC SHOWP COLDC
*CCEND

 CLB, BUILD SHOWC

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 4002  000C
 *IBT TABLE 400E  001D
 *FIO TABLE 402B  0010
 *ETV TABLE 403B  000F
 *VTV TABLE 404A  0027
 *IST TABLE 4071  0036
 *PNT TABLE 40A8  0008
 MAIN SHOWP 416F
 PNT  SHOWC 40AA
 PNT  COLDC 40AE
 CALL ENT   41FF
 LIBF SUBSC 421E  404A
 LIBF ISTOX 424A  404D
 CALL IABS  426A
 LIBF MWRT  440E  4050
 LIBF MIOI  44C4  4053
 LIBF MIOIX 44D0  4056
 LIBF MCOMP 449B  4059
 CALL VIAQ  48C0
 LIBF SUBIN 4920  405C
 LIBF COMGO 495A  405F
 LIBF ADRCK 49AC  4062
 LIBF IOU   49FE  4065
 CALL IOFIX 4AB6
 CALL BT1BT 4AE6
 CALL SAVE  4A52
 LIBF FLOAT 4B4A  4068
 LIBF IFIX  4B66  406B
 LIBF NORM  4B92  406E
 CORE       4BC0  3440

 CLB, SHOWC LD XQ

DUP FUNCTION COMPLETED
*DUMPLET  F



        FLET

PACK LABEL
 00000

.FIOS 001B  03A0    /EPDM 7FFF  03BB    /EPSV 0780  0422    /INSV 2280  0428    /NPSV 4000  0444    .MESS 0010  0478
DUMMY 0092  0488    DUMIN 005A  0489    NONPR 00F0  048A    NP    0098  048B    COLDC 0A8C  048C    WAITC 0A28  0495
READC 0DBC  049E    CALCC 0B22  04A9    SHOWC 0BBE  04B2    9DUMY 00BC  04BC    /SPSV 4000  0578    /PRSV 4000  05AC
.SKEL 0038  05E0    .EPRG 0022  0618    /CLST 0780  063A    .E    00F0  0488

DUP FUNCTION COMPLETED
```

```
     ENTERED C/L  1   EFTA
     EXITED  C/L  1   EFTA
     ENTERED C/L  2   EFTA
     EXITED  C/L  2   EFTA
```

```
// JOB
// XEQ NPJOB
*CCEND
```

Time-Sharing begins here.

```
 CLB, BUILD NPJOB

     ENTERED C/L  3   EFTA
 ICNT=  1
     EXITED  C/L  3   EFTA
     ENTERED C/L  3   EFTA
 ICNT=  2
     EXITED  C/L  3   EFTA
 CLB, NPJOB LD XQ
```

Interrupt core load on level 07 takes precedence over nonprocess job. Programmed interrupt level 07 initiated from in-skeleton timer routine called SCAN.

```
        1
        2
        3
        4
        5
        6
        7
        8
        9
       10
       11
       12
       13
       14
       15
       16
       17
       18
       19
       20
     ENTERED C/L  3   EFTA
 ICNT=  3
     EXITED  C/L  3   EFTA
       21
       22
       23
       24
       25
       26
       27
       28
       29
       30
       31
       32
       33
       34
       35
       36
       37
       38
       39
```

During time-sharing, a nonprocess job is executed and prints out a pattern of numbers in an increasing order of magnitude, as shown. This list of numbers is interrupted by core loads (mainline process or interrupt process) at a higher level.

Third entry of core load READC calls end time-sharing Time-sharing terminates the next time timer C interrupts.

```
     ENTERED C/L  4   EFTA
READINGS        9000     12000     15000
     9000       9120      9240      9360      9480      9600      9720      9840      9960     10080     10200     10320
    10440      10560     10680     10800     10920     11040     11160     11280     11400     11520     11640     11760
    11880      12000     12120     12240     12360     12480     12600     12720     12840     12960     13080     13200
    13320      13440     13560     13680     13800     13920     14040     14160     14280     14400     14520     14640
    14760      14880     15000
     EXITED  C/L  4   EFTA
     ENTERED C/L  5   EFTA
     ENTERED C/L  3   EFTA
ICNT=  1
     EXITED  C/L  3   EFTA
1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Core load 4 is executed from the QUEUE.

Core load 5 is executed from the QUEUE.

Core load 5 (SHOWC) prints histogram.

```
  6  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  7  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  8  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  9  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 11  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 12  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 13  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 14  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 15  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 16  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 17  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 18  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 19  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 20  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 21  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 22  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 23  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 24  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 25  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 26  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 27  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 28  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 29  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 30  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 31  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 32  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 33  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 34  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 35  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 36  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     ENTERED C/L   3    EFTA
 ICNT=  2
     EXITED  C/L   3    EFTA
```

During the printing of the histogram, interrupt core load READC is brought into core and executed.

```
 37  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 38  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 39  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 40  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 41  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 42  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 43  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 44  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 45  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 46  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 47  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 48  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 49  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 50  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 51  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     EXITED  C/L   5    EFTA
      40
      41
      42
      43
      44
      45
      46
      47
      48
      49
      50
      51
      52
     ENTERED C/L   3    EFTA
 ICNT=  3
     EXITED  C/L   3    EFTA
      53
      54
      55
      56
      57
      58
      59
      60
      61
      62
      63
      64
      65
      66
      67
      68
      69
      70
      71
      72
      73
      74
      75
      76
      77
      78
```

The cycle of events repeats itself.

USE OF THE OPERATIONS MONITOR

The Operations Monitor is an optional watch-dog type timer device which warns the user when the processor-controller is not executing a predicted sequence of instructions. This may be caused by power failure, computer hang-up, or computer looping.

The watch-dog timer works on the principle that a contact closes upon completion of a preset time-out period. When this occurs, a separately-powered alarm or indicator (supplied by the user) is brought into play. The time-out period is settable within the range 5-30 seconds. Note that the time interval selected must be greater than the secondary time base specified by the Interval Timer Control (ITC) program.

The user may also exercise the option of manual or automatic reset of the Operations Monitor. This option is specified in the OPMON equate card at system generation time (see System Design Considerations: System Director). Automatic resetting is undertaken by ITC during time-sharing operations. If the Operations Monitor is used, it is the user's responsibility to ensure that a reset (XIO) instruction is executed frequently enough in his program so as to prevent timeout during normal operation. If the reset command is not given during the selected interval, timeout occurs and the alarm circuit is closed.

The Call Operations Monitor subroutine is used to reset the monitor. Its format is:

```
CALL OPMON
```

Consider the following example. A particular program (say, a logging program) has been designed for execution every 15 seconds, and therefore ideally suited for Operations Monitor reset. If the program is not, for some reason, executed within this allowed time span, the Operations Monitor is set, causing an alarm in the warning device the user has attached to the Operations Monitor.

ERROR ALERT CONTROL

Error procedures in the IBM 1800 Time-Sharing Executive System are provided by a program package called the Error Alert Control (EAC) Program which is designed to analyze errors that are:

1. Basic to the hardware, and
2. which may result from incorrect use of software programs.

Since errors affect all real-time systems, from the largest to the smallest, the policy adopted towards all errors is to keep the system on-line if at all possible, and to minimize operator decisions.

Features of EAC

Error Conditions Serviced

The Error Alert Control program provides error recovery for the following conditions:

- An input/output error which persists despite repeated corrective action by an I/O subroutine.
- Occurrence of an internal machine error (e.g., invalid operation code, parity, storage protect violation)
- Other control subroutine error conditions (e.g., QUEUE, FORTRAN I/O)

Error Analysis Provisions

Provision is also made for the following features.

Dump of Core Storage to Disk. An optional dump of all core storage to disk is provided if this option is elected through the System Director equate card DUMP1 at system generation time. If, for example, DUMP1 is equated to 1, the DUMP routine is included (at System Director assembly time) with the EAC program package. This feature is only applicable to subroutine type errors.

The DUMP routine writes core into the EDP DUMP AREA on disk. Since permanent core may be storage protected, and the disk routine must insert the sector address at the start of each sector to be written, the dump routine moves blocks of six sectors of permanent core to variable core and copies it to disk. After all of permanent core has been copied, that portion of variable core used is restored.

The copied data on disk can now be dumped to an output device by the DUP *DUMP function.

User Error Subroutine. In a process program, EAC branches to a user-written error subroutine if this is included with the mainline core load. This action is bypassed for internal machine errors, if an error subroutine is not included and if a nonprocess program is in core.

A user-written error subroutine can be optionally included with each process core load. The purpose of this subroutine is to allow the user to have control before EAC overlays the variable area with the disk portion of EAC. For example, there may be special data or other information that the user wants to save. Output, such as special core dumps, messages, or contact operate functions, can also be executed. The error subroutine cannot be written in FORTRAN language.

Before entering the user's error subroutine, error identification data is placed in words 00115-00119. These words will contain the following:

Input/Output Errors

00115	Error type code
00116	Address of illegal call or address of the device table
00117	Address of level work area
00118	Address of originating call

Queue Overflow

00115	Error type code
00116	Word count of core load named in CALL QUEUE
00117	Sector address of core load named in CALL QUEUE
00118	Priority of core load named in CALL QUEUE
00119	Error parameter of core load named in CALL QUEUE

The meaning of on-line EAC error type codes is given in Table 3. Table 4 contains a description of all on-line errors serviced by EAC, the format of each EAC message printout, and corrective action specifications.

A standard recovery procedure is executed by EAC according to the type of error (see Table 4). User options are specified in the same table (see USER OPTION column). However, under certain conditions, EAC overrides the user option. The EAC option is always executed if an error subroutine is not used or the user does not specify an option. Options can be specified by the user before returning to EAC by loading the A-register with -10 for S (RESTART) or -1 for I & R (CONTINUE).

The last logical statement in the error subroutine must be a BSC I entry to the error subroutine.

The core load named for the restart option can be an error analysis core load, or it can be the first of a new series of core loads. If queueing techniques are used, the restart core load can be simply a CALL VIAQ statement (CALL QUEUE can be executed in the restart core load or the error subroutine).

The statements listed below cannot be used in an error subroutine:

CALL BACK	CALL QIFON
CALL CHAIN	CALL RESMK
CALL DPART	CALL SAVMK
CALL ENDTS	CALL SHARE
CALL EXIT	CALL SPECL
CALL INTEX	CALL UNMK
CALL LEVEL	CALL VIAQ
CALL LINK	
CALL MASK	

Update Error Counters Maintained on Disk. For each I/O unit on the system, a hardware counter is maintained on the disk for printout to the Customer Engineer for maintenance purposes.

Back-up Capability for D.P. I/O Units. The option of including backup units for the 1053 and the 1816, as well as the logical removal of the 1443 from service, can be specified at system generation time. If backup is not provided, the 1053 printer will be automatically removed from service when multiple failures occur without taking the system off-line.

Backup for the EAC printer is achieved by defining multiple EAC printers at TASK assembly time (if the EAC printer is defined as a 1053). When an output error occurs, or if the unit is not ready (that is, interrupt response is not received), EAC will logically disconnect the unit in error and substitute the backup unit. When backup is initiated because of a hardware malfunction, the message in progress on the failing unit is not continued on the backup device. When the error condition is corrected, the unit can be restored to its original status by using the C. E. Interrupt routine. See C. E. Interrupt Routine in the publication IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

EAC Program Breakdown

EAC can be considered in terms of four component parts; each component functions as a separate subprogram, the four parts remaining interdependent insofar as the status information of the error (detected) is shared by all routines concerned. In addition, EAC sets up a level work area for the use of reentrant coded programs when it is processing

Table 3. On-Line EAC Error Type Codes

EAC MESSAGE FORMAT

*INN CL.OCK AC-M PNAME LOCN

*	INDICATES PROCESS CORELOAD IN CORE
BLANK	INDICATES NON-PROCESS CORELOAD IN CORE
I	GENERAL I/O
P	PROCESS I/O
F	FORTRAN
Q	QUEUE
M	MASK
X	MISCELLANEOUS
NN	TWO DIGIT NUMBER INDICATING TYPE OF ERROR
CL.OCK	TIME IN THOUSANDTHS OF AN HOUR
AC	AREA CODE FOR THE ASSOCIATED I/O DEVICE
M	MODIFIER IF MORE THAN ONE FOR THAT AREA CODE
PNAME	NAME OF THE PROGRAM IN CORE AT THE TIME OF THE MESSAGE (NOT NECESSARILY THE ONE WHICH ORIGINATED THE CALL LEADING TO THE ERROR CONDITION)
LOCN	LOCATION OF THE CALL

USER ERROR TYPE CODES FOR DP I/O

I01	PARITY
I02	STORAGE PROTECT
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
I16	END OF TAPE
I17	INVALID CALL TO ERROR ROUTINE
I18	NO RESPONSE FROM DISK
I19	INVALID DISK ADDRESS

USER ERROR TYPE CODES FOR PROCESS I/O

P01	PARITY DATA OR COMMAND REJECT
P02	STORAGE PROTECT VIOLATION
P03	ILLEGAL CALL
P04	PARITY CONTROL
P05	OVERLAP CONFLICT
P17	INVALID ERROR CODE

USER ERROR TYPE CODES FOR QUEUING

Q01	ERROR OPTION IS ZERO - CALL IGNORED
Q02	ERROR OPTION NOT ZERO - NO LOWER PRIORITY IN QUEUE
Q03	QUEUE ENTRY REPLACED BY NEW CALL QUEUE
Q04	QUEUE CALL NOT HONORED - RESTART INITIATED
Q17	INVALID ERROR CODE

USER ERROR TYPE CODES FOR FORTRAN

F90	ILLEGAL ADDR COMPUTED IN AN INDEXED STORE
F91	ILLEGAL INT USED IN A COMPUTED GO TO

USER ERROR TYPE CODES FOR FORTRAN (CONTINUED)

	DISK I/O
F92	FILE NOT DEFINED
F93	RECORD TOO LARGE. ZERO OR NEGATIVE
	NON-DISK I/O
F94	INPUT RECORD IS IN ERROR
F95	RANGE OF NUMERICAL VALUES IS IN ERROR
F96	OUTPUT FIELD TOO SMALL TO CONTAIN THE NUMBERS
F97	ILLEGAL UNIT REFERENCE
F98	REQUESTED RECORD EXCEEDS ALLOCATED BUFFER
F99	WORKING STORAGE AREA INSUFFICIENT FOR DEFINED FILES
F17	INVALID ERROR CODE
	UNEDITED I/O
F87	ILLEGAL UNIT REFERENCE
F88	READ LIST EXCEEDS LENGTH OF WRITE LIST
F89	RECORD DOES NOT EXIST FOR READ LIST ELEMENT

USER ERROR TYPE CODES FOR MASK ROUTINES

M01	ILLEGAL CALL RESMK
M02	ILLEGAL CALL UNMK
M17	INVALID ERROR CODE

USER ERROR TYPE CODES FOR PROGRAM SEQUENCE CONTROL

X01	ILLEGAL CALL BACK
X02	INTERRUPT LEVEL ERROR
X03	CORELOAD NOT LOADED ON DISK
X04	RESTART CORELOAD NOT LOADED ON DISK
X17	INVALID ERROR CODE

Table 4. On-Line EAC Errors and Recovery Procedures

ERROR CODE DEC	ERROR CODE HEX	TYPE CODE EAC	EAC STAND. EXIT	USER OPTION	ERROR MESSAGE AND COMMENTS
1053/1816 PRINTER/KEYBOARD					
00	00	103	S	N	ILLEGAL CALL 103 CL.OCK PNAME LOCN 1053 USER MUST CORRECT CALL IN PROGRAM
01	01	104	R,S	S*	1053 NOT READY 104 CL.OCK AC-M PNAME 1053 NOT READY CHECK FORMS
03	03	104	R,S	R,S	1816 KEYBOARD NOT READY 104 CL.OCK AC-M PNAME 1816 NOT READY MAKE READY
04	04	102	L	N	STORAGE PROTECT VIOLATION FROM 1816 102 CL.OCK AC-M PNAME 0000 USER MUST CHECK PROGRAM
05	05	101	S	R	KEYBOARD PARITY ERROR 101 CL.OCK AC-M PNAME 1816 PARITY LAST CHARACTER TYPED MAY BE INVALID
06	06	101	I	N*	PRINTER PARITY ERROR 101 CL.OCK AC-M PNAME 1053 PARITY AN ATTEMPT TO PRINT HAS BEEN MADE 2 TIMES
07	07	110	R	N*	NO PRINT RESPONSE 101 CL.OCK AC-M PNAME NO PRINT RESP NO OP COMPLETE HAS BEEN RECEIVED
08	08	112	R	N	INVALID MESSAGE ON DISK 112 CL.OCK AC-M PNAME THIS MESSAGE IS NOW LOST
1442 CARD READ-PUNCH					
10	0A	103	S	N	ILLEGAL CALL TO 1442 103 CL.OCK PNAME LOCN 1442 USER MUST CORRECT CALL IN PROGRAM
11	0B				LAST CARD
12	0C	101	R	S	PARITY ERROR 101 CL.OCK AC PNAME 0000 1442 PARITY NON-PROCESS RUN OUT, RELOAD UN-READ CARDS
13	0D	102	L	N	STORAGE PROTECT VIOLATION 102 CL.OCK AC PNAME 0000 USER MUST CHECK PROGRAM
14	0E	106	R	S	FEED CHECK 106 CL.OCK AC PNAME 1442 NOT READY NON-PROCESS RUN OUT, RELOAD UN-READ CARDS
15	0F	108			DATA OVERRUN 108 CL.OCK AC PNAME 0000 1442 NOT READY NON-PROCESS RUN OUT, RELOAD UN-READ CARDS
16	10	107	R	S	READ-PUNCH CHECK 107 CL.OCK AC PNAME 1442 NOT READY NON-PROCESS RUN OUT, RELOAD UN-READ CARDS
17	11	105	S	N	//BLANK CARD 105 CL.OCK AC PNAME 0000 CONTROL CARD ENCOUNTERED - CHECK DECK
19	13	104	R	S	1442 NOT READY 104 CL.OCK AC PNAME 1442 NOT READY PRESS START ON UNIT
1054/1055 PAPER TAPE READER/PUNCH					
20	14	103	S	N	ILLEGAL CALL 103 CL.OCK PNAME LOCN 1054 USER MUST CORRECT CALL IN PROGRAM
21	15	101	S	I	PUNCH PARITY ERROR 101 CL.OCK AC PNAME 0000 1055 PARITY LAST CHARACTER OUT MAY BE INVALID
22	16	104	R,S	S	READER NOT READY 104 CL.OCK AC PNAME 1054 NOT READY MAKE READY
23	17	104	R,S	S	PUNCH NOT READY 104 CL.OCK AC PNAME 1055 NOT READY MAKE READY

LEGEND FOR EAC STANDARD EXIT AND USER OPTION:

I	- CONTINUE AT THE POINT OF INTERRUPT
R	- RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S	- RESTART
L	- RELOAD
N	- NO OPTION - MUST TAKE EAC EXIT
*	- INTERNAL BACKUP ATTEMPTED

Table 4. On-Line EAC Errors and Recovery Procedures

ERROR CODE DEC	ERROR CODE HEX	TYPE CODE EAC	EAC STAND. EXIT	USER OPTION	ERROR MESSAGE AND COMMENTS
1054/1055 PAPER TAPE READER/PUNCH (Cont'd)					
24	18	101	S	I	READER PARITY ERROR 101 CL.OCK AC PNAME 0000 1054 PARITY LAST CHARACTER READ IN MAY BE IN ERROR
25	19	102	L	N	READER STORAGE PROTECT 102 CL.OCK AC PNAME 0000 USER MUST CHECK HIS PROGRAM FOR ERROR(S)
2310 DISK					
30	1E	103	S	N	ILLEGAL CALL 103 CL.OCK PNAME LOCN 2310 USER MUST CORRECT CALLING SEQUENCE
31	1F	104	R	S	DISK NOT READY 104 CL.OCK AC PNAME 2310 NOT READY MAKE READY
32	20	108	S	I	DATA OVERRUN 108 CL.OCK AC PNAME 0000 2310 HARDWARE ERROR INVALID DATA FROM DISK AFTER 10 TRIES
33	21	109	S	I	WRITE SELECT 109 CL.OCK AC PNAME 0000 2310 HARDWARE ERROR STOP DISK AND START AGAIN TO RESET
34	22	111	S	I	DATA ERROR 111 CL.OCK AC PNAME 0000 2310 HARDWARE ERROR EXCESSIVE WD CT FOR SECTOR OR MODULO 4 ERROR
35	23	102	L	N	STORAGE PROTECT ERROR 102 CL.OCK AC PNAME 0000 USER MUST CHECK HIS PROGRAM FOR ERROR(S)
36	24	101	S	I	PARITY ERROR 101 CL.OCK AC PNAME 0000 2310 HARDWARE ERROR ERROR PERSISTS AFTER 10 TRIES
37	25	119	S	N	INVALID DISK ADDRESS 119 CL.OCK AC PNAME 0000 INVALID ADDRESS OR UNEXPECTED HOME BIT ON
38	26	113	S	N	FILE PROTECT ERROR 113 CL.OCK AC PNAME 0000 USER TRIED WRITING IN A FILE PROTECTED SECTOR
39	27	118	S	N	NO RESPONSE 118 CL.OCK AC PNAME 0000 2310 HARDWARE ERROR DID NOT RECEIVE OR LOST RESPONSE FROM DISK
1627 PLOTTER					
41	29	101	S	I	PARITY ERROR 101 CL.OCK AC PNAME 0000 1627 PARITY NO ATTEMPT IS MADE TO REPLOT THE POINT
42	2A	104	R,S	S	NOT READY 104 CL.OCK AC PNAME 1627 NOT READY MAKE READY
1443 PRINTER					
50	32	103	S	N	ILLEGAL CALL 103 CL.OCK PNAME LOCN 1443 USER MUST CORRECT CALL IN PROGRAM
53	35	110	R,S	R,S	NO PRINT RESPONSE 110 CL.OCK AC PNAME 1443 NOT READY PUSH START ON THE PRINTER
54	36	101	S,I	I	PARITY ERROR 101 CL.OCK AC PNAME 0000 1443 PARITY NO ATTEMPT IS MADE TO REPRINT THE LINE
55	37	104	R,S	R,S	NOT READY 104 CL.OCK AC PNAME 1443 NOT READY PUSH RESET AFTER CORRECTING PRINTER ERROR THEN PUSH START

LEGEND FOR EAC STANDARD EXIT AND USER OPTION:

I - CONTINUE AT THE POINT OF INTERRUPT
R - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S - RESTART
L - RELOAD
N - NO OPTION - MUST TAKE EAC EXIT
* - INTERNAL BACKUP ATTEMPTED

(Continued)

Table 4. On-Line EAC Errors and Recovery Procedures

ERROR CODE DEC	HEX	TYPE CODE EAC	EAC STAND. EXIT	USER OPTION	ERROR MESSAGE AND COMMENTS
ANALOG INPUT BASIC					
60	3C	P03	S	N	ILLEGAL CALL P03 CL.OCK PNAME LOCN AIN ILLEGAL CALL SEQUENCE IN PROGRAM
61	3D	P02	L	N	STORAGE PROTECT VIOLATION P02 CL.OCK AC PNAME 0000 AIN WRITE INTO MEMORY PROTECTED LOCN ATTEMPTED
62	3E	P04	S	N	PARITY CONTROL ERROR P04 CL.OCK AC PNAME 0000 AIN PARITY ERROR ON DATA OR CONTROL CYCLE
63	3F	P01	S	N	PARITY DATA ERROR P01 CL.OCK AC PNAME 0000 AIN PARITY ERROR DURING TRANSMISSION
64	40	P05	S	N	OVERLAP CONFLICT P05 CL.OCK AC PNAME 0000 AIN RELAY POINTS IN RANDOM READ FUNCTION TOO CLOSE TOGETHER
65	41		S	N	INTERMEDIATE INTERRUPT DATA TRANSFER COMPLETED DURING CHAIN OPERATION
66-68					NOT USED
		P17	S	N	INVALID ERROR CODE P17 CL.OCK PNAME AIN INVALID ERROR CODE FROM EAC
DIGITAL INPUT BASIC					
70	46	P03	S	N	ILLEGAL CALL P03 CL.OCK PNAME LOCN DIN ILLEGAL CALL SEQUENCE IN PROGRAM
71	47	P01	S	N	PARITY ERROR OR COMMAND REJECT P01 CL.OCK AC PNAME 0000 DIN DATA TRANSMITTED INCORRECTLY OR ILL. REQUEST
72	48	P02	L	N	STORAGE PROTECT ERROR P02 CL.OCK AC PNAME 0000 DIN WRITE OPERATION TRIED IN MEMORY PROTECTED LOCN
73	49		S	N	INTERMEDIATE INTERRUPT DATA TRANSFER COMPLETED DURING CHAIN OPERATION
74-79					NOT USED
		P17	S	N	INVALID ERROR CODE P17 CL.OCK PNAME DIN INVALID ERROR CODE FROM EAC
DIGITAL AND ANALOG OUTPUT BASIC					
80	50	P03	S	N	ILLEGAL CALL P03 CL.OCK PNAME LOCN DAO ILLEGAL CALLING SEQUENCE IN PROGRAM
81	51	P01	S	N	PARITY ERROR OR COMMAND REJECT P01 CL.OCK AC PNAME 0000 DAO DATA TRANSMITTED INCORRECTLY OR ILL. REQUEST
82	52		S	N	INTERMEDIATE INTERRUPT DATA TRANSFER COMPLETED DURING CHAIN OPERATION
83-89					NOT USED
		P17	S	N	INVALID ERROR CODE P17 CL.OCK PNAME DAO INVALID ERROR CODE FROM EAC
2402 MAG TAPE					
90	5A	I03	S	N	ILLEGAL CALL I03 CL.OCK PNAME LOCN 2402 ILLEGAL CALL SEQUENCE IN PROGRAM
91					NOT USED
92	5C	I02	L	N	STORAGE PROTECT VIOLATION I02 CL.OCK AC PNAME 0000 2402 WRITE INTO MEMORY PROTECTED LOCN ATTEMPTED

LEGEND FOR EAC STANDARD EXIT AND USER OPTION:

I	- CONTINUE AT THE POINT OF INTERRUPT
R	- RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S	- RESTART
L	- RELOAD
N	- NO OPTION - MUST TAKE EAC EXIT
*	- INTERNAL BACKUP ATTEMPTED

(Continued)

Table 4. On-Line EAC Errors and Recovery Procedures

ERROR CODE DEC	ERROR CODE HEX	TYPE CODE EAC	EAC STAND. EXIT	USER OPTION	ERROR MESSAGE AND COMMENTS
2402 MAG TAPE (Cont'd)					
93	5D	113	S	R	COMMAND REJECT 113 CL.OCK AC PNAME 0000 2402-COMMAND REJ ILL MT OPERATION REQUESTED. USER CHECK PROGRAM
94	5E	115	S	R	EXCESSIVE TAPE ERRORS 115 CL.OCK AC PNAME 0000 2402-EXCESS ERR TOO MANY FAILS ON THIS REEL. MOUNT NEW REEL
95	5F	114	R	S	TAPE ERROR 114 CL.OCK AC PNAME 0000 2402-TAPE ERR DSW DSW- DEVICE STATUS WORD PARITY ERROR OR OTHER FAIL CONDITION AFTER 100 READ ATTEMPTS OR 3 WRITE ATTEMPTS
96-97					NOT USED
98	62	104	R,S	S	NOT READY 104 CL.OCK AC-M PNAME 2402-NOT READY MAKE READY
99	63	116	R	S	END OF TAPE 116 CL.OCK AC PNAME 0000 2402-END OF TAPE OPERATION ATTEMPTED PAST END OF TAPE
		117			INVALID ERROR CODE 117 CL.OCK PNAME MAG INVALID ERROR CODE FROM EAC
FORTRAN					
100	64	F90	S	N	ILLEGAL ADDR COMPUTED IN AN INDEXED STORE SUBSCRIPTED VALUE OUTSIDE LIMITS OF ARRAY F90 CL.OCK PNAME LOCN
101	65	F91	S	N	ILLEGAL INTEGER VALUE IN COMPUTED GO TO F91 CL.OCK PNAME LOCN
					DISK I/O
102	66	F92	S	N	FILE NOT DEFINED F92 CL.OCK PNAME LOCN FILE REQUESTED NOT DEFINED IN DEFINE FILE STATEMENT
103	67	F93	S	N	REQUESTED NO. OF RECORDS TOO LARGE, ZERO, OR NEGATIVE F93 CL.OCK PNAME LOCN
					NON-DISK I/O
104	68	F94	S	N	INPUT RECORD IN ERROR F94 CL.OCK PNAME LOCN ILLEGAL CHARACTER IN NUMERIC FIELD OR ILLEGAL CONVERSION
105	69	F95	S	N	RANGE OF NUMERICAL VALUES IS IN ERROR F95 CL.OCK PNAME LOCN FIXED OR FLOATING PT NUMBER OUTSIDE DEFINED LIMITS
106	6A	F96	R	N	REQUESTED OUTPUT FIELD TOO SMALL F96 CL.OCK PNAME LOCN
107	6B	F97	S	N	ILLEGAL UNIT REFERENCE F97 CL.OCK PNAME LOCN UNIT NOT DEFINED IN IOU TABLE OR IOCS CONTROL CARD
108	6C	F98	S	N	REQUESTED RECORD EXCEEDS ALLOCATED BUFFER F98 CL.OCK PNAME LOCN RECORD SIZE TOO LARGE
109	6D	F99	S	N	WORKING STORAGE AREA INSUFFICIENT FOR DEFINE FILES F99 CL.OCK PNAME LOCN
		F17			INVALID ERROR CODE F17 CL.OCK PNAME FOR INVALID ERROR CODE FROM EAC
					UNFORMATED I/O
150	96	F87	S	N	ILLEGAL UNIT REFERENCE F87 CL.OCK PNAME LOCN UNIT NOT DEFINED IN IOU TABLE, ON IOCS CARD, OR FOR UNFORMATED I/O

LEGEND FOR EAC STANDARD EXIT AND USER OPTION:

I - CONTINUE AT THE POINT OF INTERRUPT
R - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S - RESTART
L - RELOAD
N - NO OPTION - MUST TAKE EAC EXIT
* - INTERNAL BACKUP ATTEMPTED

(Continued)

Table 4. On-Line EAC Errors and Recovery Procedures

ERROR CODE DEC	HEX	TYPE CODE EAC	EAC STAND. EXIT	USER OPTION	ERROR MESSAGE AND COMMENTS
FORTRAN (Cont'd)					
151	97	F88	S	N	READ LIST EXCEEDS LENGTH OF WRITE LIST F88 CL.OCK PNAME LOCN LIST IN READ STATEMENT IS LONGER THAN LIST IN CORRESPONDING WRITE STATEMENT
152	98	F89	S	N	RECORD DOES NOT EXIST FOR READ LIST ELEMENT F89 CL.OCK PNAME LOCN LAST PHYSICAL RECORD OF LOGICAL RECORD HAS BEEN EXHAUSTED
MISCELLANEOUS					
110	6E	X01	S	N	PSC CALL BACK ERROR X01 CL.OCK PNAME LOCN CALL BACK TRIED BEFORE CALL SPECIAL
111	6F	X03	S	N	CORELOAD NOT LOADED ON DISK X03 CL.OCK PNAME COREN COREN - CORELOAD NOT LOADED
112	70	X04	L	N	RESTART CORELOAD NOT LOADED ON DISK X04 CL.OCK PNAME COREN COREN - CORELOAD NOT LOADED
		X17			INVALID ERROR CODE X17 CL.OCK PNAME CLB INVALID ERROR CODE FROM EAC
120	78	Q01	S	N	QUEUE CALL IGNORED ERROR OPTION ZERO Q01 CL.OCK WC SA P WC- 5 DIGIT WORD COUNT SA - 5 DIGIT SECTOR ADDRESS P - 5 DIGIT PRIORITY
120	78	Q02			QUEUE CALL NOT HONORED-NO LOWER PRIORITY IN QUEUE ERR OPTION 1 TO 32766 Q02 CL.OCK WC SA P
120	78	Q03			QUEUE CALL HONORED-CALL ENTERED IN QUEUE ERR OPTION 1 TO 32766 Q03 CL.OCK WC SA P REPLACES WC SA P
120	78	Q04			QUEUE CALL NOT HONORED-RESTART INITIATED ERR OPTION 32767 Q04 CL.OCK WC SA P
		Q17			INVALID ERROR CODE Q17 CL.OCK PNAME QUE INVALID ERROR CODE FROM EAC
130	82	M01	S	N	CALL RESMK ERROR M01 CL.OCK PNAME LOCN ATTEMPT TO UNMASK OUT OF CORE INTERRUPT LEVEL WHILE IN AN OUT OF CORE INTERRUPT PROGRAM
131	83	M02	S	N	CALL UNMK ERROR M02 CL.OCK PNAME LOCN ATTEMPT TO UNMASK OUT OF CORE INTERRUPT LEVEL WHILE IN AN OUT OF CORE INTERRUPT PROGRAM
		M17			INVALID ERROR CODE M17 CL.OCK PNAME MSK INVALID ERROR CODE FROM EAC
140	8C	X02	S	N	INTERRUPT LEVEL ERROR X02 CL.OCK PNAME LOCN ATTEMPT TO CALL LEVEL UNDEFINED FOR SYSTEM
		X17			INVALID ERROR CODE X17 CL.OCK PNAME LEV INVALID ERROR CODE FROM EAC

LEGEND FOR EAC STANDARD EXIT AND USER OPTION:

I - CONTINUE AT THE POINT OF INTERRUPT
R - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S - RESTART
L - RELOAD
N - NO OPTION - MUST TAKE EAC EXIT
* - INTERNAL BACKUP ATTEMPTED

(Continued)

Table 4. On-Line EAC Errors and Recovery Procedures

```
INTERNAL ERRORS

          CAR CHECK ERROR
          996 CL.OCK PNAME OPTION
                 SKELETON - RELOAD
                 VARIABLE  - RESTART

          OP CODE VIOLATION
          997 CL.OCK PNAME OPTION
                 SKELETON - RELOAD
                 VARIABLE  - RESTART

          STORAGE PROTECT VIOLATION
          998 CL.OCK PNAME OPTION
                 SKELETON - RELOAD
                 VARIABLE  - RELOAD

          PARITY ERROR
          999 CL.OCK PNAME OPTION
                 SKELETON - RELOAD
                 VARIABLE  - RESTART

          OPTION WILL BE RELOAD (IF ERROR IS IN
          SKELETON), RESTART (VARIABLE CORE - ABORT OF
          NONPROCESS JOB, OR USER'S RESTART CORE LOAD
          IF PROCESS), OR COLD START (REQUIRED IF EAC
          IS UNABLE TO RELOAD SYSTEM)

          MULTIPLE ENTRANCE TO EAC
          MLPT EAC

          AN ERROR HAS OCCURRED WHILE EAC WAS PROCESSING
          A PREVIOUS ERROR.  MUST GO TO A COLD START.

          NORMALLY THIS ERROR INDICATES THAT THE
          SYSTEM DISK IS DOWN.  THIS ERROR WILL ALSO
          OCCUR IF AN ERROR OCCURS IN EAC WHILE EAC
          IS ATTEMPTING TO PROCESS A SYSTEM ERROR.
```

LEGEND FOR EAC STANDARD EXIT AND USER OPTION:

I	- CONTINUE AT THE POINT OF INTERRUPT
R	- RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S	- RESTART
L	- RELOAD
N	- NO OPTION - MUST TAKE EAC EXIT
*	- INTERNAL BACKUP ATTEMPTED

(Continued)

an error. Note that the user cannot call from his error subroutines any routine that utilizes more than 14 words of the subroutine work area (a portion of the level work area). This area is principally used for those calls to disk and output printers used by EAC. It may be increased in size if the user elects to remove this restriction.

The EAC program is entered whenever an error occurs or a condition arises that calls for operator intervention. An error message is then given on the EAC printer and the program takes one of five possible exits after proper analysis has determined which exit may be taken for the error in question. Where more than one exit pertains to a given error condition, the user has the option of specifying the exit desired from his (user) error subroutine.

The four component parts are described below.

EAC In-Core. The in-core component of EAC is an integral part of the System Director and resides in core storage at all times. Its main function is to channel one of the several possible types of errors to a specific entry such that information relating to this particular error is passed on correctly to the analysis section. It also saves the current machine status so that after an error has been processed, the exit routine can return the machine back to the user without loss of information. EAC in-core also has the ability to dump variable core to disk if this is specified by the user at System Director assembly time (see System Design Considerations: System Director). This program also determines conditions such as process or nonprocess mode, invalid operation code, parity errors, and user error subroutine availability.

Error Disk Program (EDP). EDP resides permanently on disk, except when it is called to core by the EAC in-core program. Once EDP is in core, it takes the error information from the fixed area and determines what type of error has arisen, the approximate address at which it occurred, and the appropriate error processing subroutine; prior to this, the correct entry addresses for the conversion and error routines are initialized. When the error processing routine has completed its task, certain information such as perform a Cold Start or Restart, or this error was not corrected but we are continuing the process, or this error has been successfully corrected, etc., are passed to the Exit component.

Error Decision Subroutines. These subroutines reside on disk at all times until called to main core by EDP to process a particular error. After the error processing has taken place, a decision is made on the type of recovery procedure required (e.g., Continue processing, Restart, Reload). This information is then passed to the Exit component of EAC for execution.

EAC Exit. This is the means by which a branch is made to the recovery exit prescribed by the Error Decision Subroutine. Note that there is no normal exit from EAC.

Action of EAC When an Error Occurs

Consider the train of events that takes place when an error occurs, as shown in the simplified block diagram, Figure 27. The error may be an Internal Machine Error, a C.E. Interrupt, or a Miscellaneous Subroutine Error which may be an error or condition requiring outside intervention. Depending on the type of error, one of three possible entries is made to EAC, as follows:

Internal Errors:	EAC00
C.E. Interrupt:	EAC01
Miscellaneous Error:	EAC02

The explanatory paragraphs that follow are given in an alphabetic sequence which corresponds exactly to blocks within Figure 27.

A. An Internal Machine Error may be the result of:

- Parity
- An invalid operation code
- A storage protect violation, or
- A Channel Address Register (CAR) check

When such an error occurs, the hardware generates a BSI indirect to EAC00 through word 8 where the processing procedure begins. The return address, the status of the accumulator and its extension, the type of error and certain registers are now saved, and the machine put in a fully masked state. For each

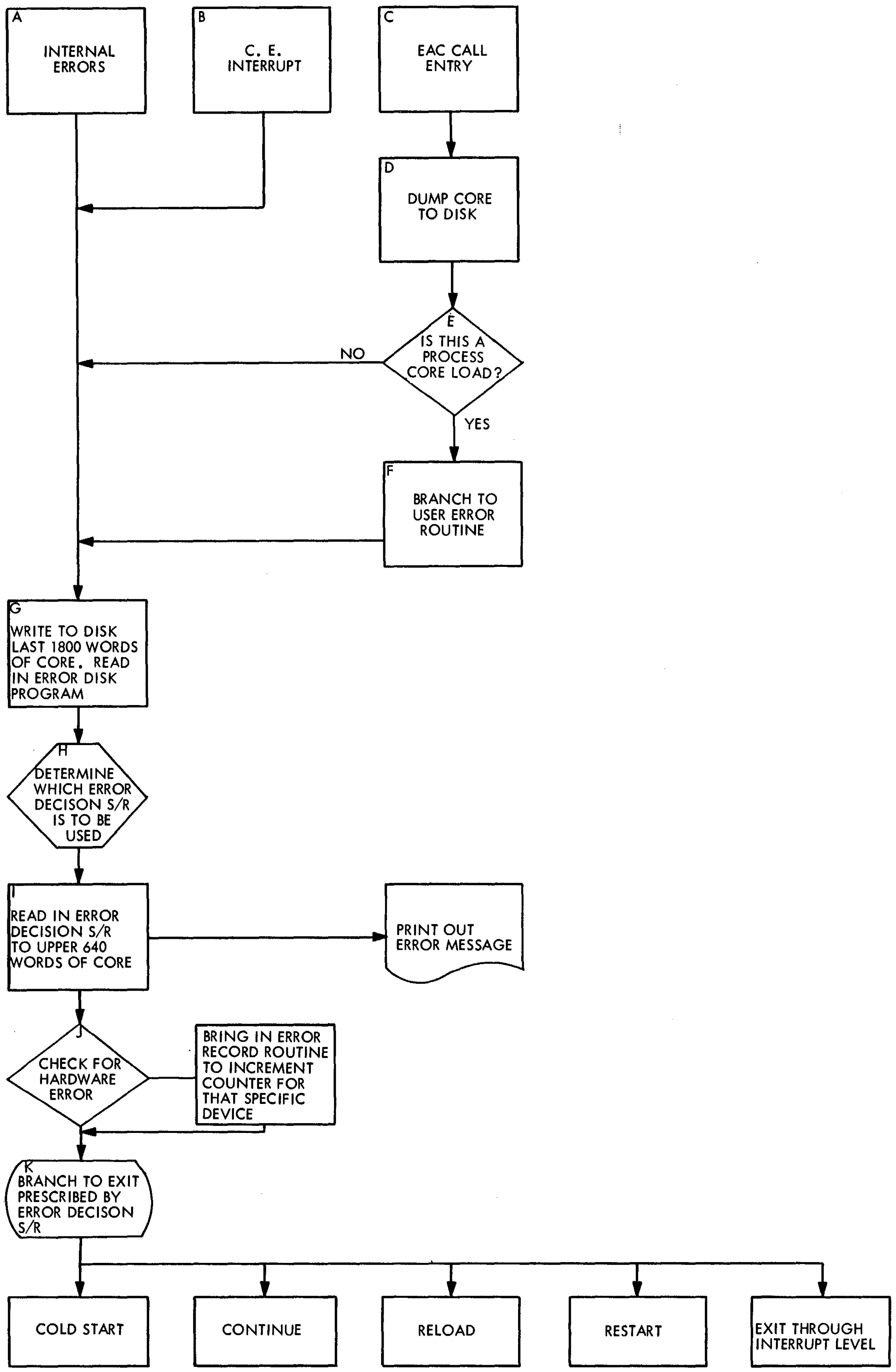

A
INTERNAL ERRORS
B
C. E. INTERRUPT
C
EAC CALL ENTRY
D
DUMP CORE TO DISK
E
IS THIS A PROCESS CORE LOAD?
NO
YES
F
BRANCH TO USER ERROR ROUTINE
G
WRITE TO DISK LAST 1800 WORDS OF CORE. READ IN ERROR DISK PROGRAM
H
DETERMINE WHICH ERROR DECISON S/R IS TO BE USED
I
READ IN ERROR DECISION S/R TO UPPER 640 WORDS OF CORE
PRINT OUT ERROR MESSAGE
J
CHECK FOR HARDWARE ERROR
BRING IN ERROR RECORD ROUTINE TO INCREMENT COUNTER FOR THAT SPECIFIC DEVICE
K
BRANCH TO EXIT PRESCRIBED BY ERROR DECISON S/R
COLD START
CONTINUE
RELOAD
RESTART
EXIT THROUGH INTERRUPT LEVEL

Figure 27. Action of EAC when an Error Occurs

error level, EAC then sets up a work area within the Fixed Area. Note that the user's error routine (if included with his core load) will be ignored on all internal machine errors and nonprocess programs. A direct branch is then made to the variable core procedure (G).

B. A C.E. Interrupt routine forms part of a TSX on-line system to allow the user to check and modify system unit assignments of 1053 output printers, 1443 printers, and 2310 disk drives, and to initiate backup procedures if and when this becomes necessary. This is normally a Customer Engineer responsibility. When the C.E. LEVEL INTERRUPT toggle switch, located on the C.E. panel, is activated, a C.E. interrupt occurs, forcing a BSI to that level, and, after some processing, another BSI is generated to an entry EAC01 in EAC. Essential information is saved exactly as for internal machine errors. A direct branch is then taken to G.

C. A Miscellaneous Error which is neither an internal machine error nor a C.E. interrupt may be either an error or a condition that requires outside intervention. For example, a not-ready condition on an I/O device has arisen. This condition has been detected by the I/O routine, which then sets up an error code and additional parameters in locations 115 through 119, and finally executes a BSI through location 120 to an entry EAC02 in EAC.

D. If the dump-core-to-disk option is specified by the user at System Director assembly time, permanent core is written to the Error Dump Area on disk for interrogation at a later date. Only the latest error is kept since there is only one Save Area.

E. A determination is now made if the core load in error is a process core load.

F. If it is, a branch is made to the user's error subroutine, if it has been included in the core load, to allow him to perform such processing as he may require for his particular system. This also permits him to modify some system options. Upon return from this routine, any indicator that may have been set is saved. If it is not a process core load, an exit is made to the common variable core procedure at G.

G. At this point, DISKN is called in from the Skeleton to write out the last $(1920)_{10}$ words of core, and to read the Error Disk Program (EDP) into this area. The EDP program is origined such that it will always reside in the final $(1920)_{10}$ words of variable core. The very last 640_{10} words of EDP is the overlay area for the appropriate Error Decision Subroutine when called.

H. Upon entering the Error Disk Program, an analysis is made to determine which Error Decision Subroutine is to be used.

I. For an EAC00 entry, the Level 0 routine is used; for a C.E. Interrupt entry, the C.E. Interrupt routine is used. If the entry was made through EAC02, the routine used will be determined by the error code stored in location 115 by the routine which determined the error.

The appropriate Error Decision Subroutine is now read into the upper $(640)_{10}$ words of core; it then builds and prints the error message on up to four output printers, as defined at TASK assembly time, and sets a predetermined exit indicator or the indicator set by the user's error subroutine. A return is made to the EDP program.

J and K. A hardware error check is now carried out. Assuming that an internal machine error had occurred, an error record routine is brought in to increment (that is, update) a counter associated with that particular piece of hardware. The record of the hardware error is kept such that when maintenance is required, the counter is output to inform the Customer Engineer how often a particular hardware device has failed. Control is then returned to EAC, and the stage set for recovery action.

As shown in Figure 27, five types of recovery action (as prescribed by the appropriate Error Decision Routine) are possible.

1. Cold Start
2. Continue
3. Reload
4. Restart, and
5. Exit through an interrupt level

Cold Start. Whenever an error occurs which cannot be corrected, EAC prints a cold start procedure message, and comes to a wait state. For example, consider a machine parity error which has occurred when one of the 18 bits of information has been lost. A parity error routine then attempts to clear the error by successively loading and storing data into the affected location. If the error persists after repeated attempts at recovery, the routine prints the location of the parity error, and comes to a wait.

Continue. The error is noted, but it is not of such a nature as to interfere with the program in progress. For example, the entry to EAC may have been a C. E. Interrupt or a request to print a message for outside intervention reflecting a not-ready state, a non-fatal error, or a printer parity.

The "continue" recovery action implies that the program proceeds at the point of interruption. Consider an I/O device which has just completed its operation -- an interrupt is generated. This will transfer control to the I/O routine which then determines the correct error condition, and branches to EAC. This exit option bypasses the I/O routine and returns control to the point in the program at which the interrupt developed.

Reload. The Reload recovery routine is brought into core whenever it is suspected that some of the non-storage protected words in permanent core may have been destroyed. The routine then saves the tables necessary for the completion of previous core loads, having first verified that these tables have not been destroyed. The Skeleton (from disk) is then read by sectors into a buffer area, comparing each word to its corresponding word in core, and overlaying it if there is a difference.

If a storage protection violation has in fact occurred, this means that a storage protected word has dropped bits, thus making it different from its corresponding word on disk. Under these circumstances, a cold start must be performed. Upon completion of the Skeleton reload, various conditions and indicators will be initialized and the routine exits by way of a CALL CHAIN.

Note that if an error has occurred outside of the Skeleton Area, the present core load is aborted and a new core load is read into main core for execution.

The CAR error may be caused by incorrect usage of the "XIO" instruction or incorrect chaining of data tables, etc. This is always a reload condition.

Restart. An error has occurred which prohibits the present core load from continuing. Three types of "restart" are used:

1. If the error, such as an illegal call, occurred in a process core load, the program in progress is aborted and its restart core load is called into core for execution.
2. If the error occurred in a nonprocess core load, the job is aborted by calling in the Nonprocess Supervisor.
3. If the error occurred in an interrupt core load, this core load is aborted, and the restart core load of the current process core load is called for execution. This means that the user's restart routines must be written in such a way as to analyze his system and determine what program will be called for execution.

Exit through an Interrupt Level. A restart condition has arisen on a level other than the mainline level. The level on which the error occurred is terminated and the Restart procedure taken when the mainline level is reached.

THE NONPROCESS MONITOR

The Nonprocess Monitor (NPM) is an independent programming system which is designed to function in one of two possible modes within a TSX system:

- In the on-line mode, it operates under the control of the System Director.
- It can also be run in the off-line mode as a dedicated monitor system under TASK control.

The user elects either system (that is, an on-line or off-line system) at system generation time (see System Design Considerations).

The NPM serves a three-fold purpose:

1. It permits better computer utilization through time-sharing.
2. It allows the user to compile, assemble, store, delete, and modify programs with extreme flexibility. Because the system programs are resident on disk, only source statements and data cards are required to be read in.
3. It provides for job stacking at the Card Reader, which is fast because less card handling is required. A stacked-job environment permits automatic and uninterrupted operation.

The primary function of the Nonprocess Monitor is to provide continuous processor-controller operation during a sequence of jobs that might otherwise involve several independent programming systems. The monitor coordinates the processor-controller activity by establishing a common communications area in core storage, which is used by the various programs that make up the monitor. It also guides the transfer of control between monitor programs and the user's programs. Operation is continuous and setup time is reduced to a minimum, thereby effecting a substantial time saving in processor-controller operation and allowing greater programming flexibility.

Figure 28 illustrates the five distinct but interdependent programs which make up the Nonprocess Monitor.

NONPROCESS SUPERVISOR (SUP)

The Nonprocess Supervisor directs and controls the execution of all nonprocess programs which may be either IBM-supplied as part of the TSX package (e.g., FORTRAN Compiler, Assembler, Core Load Builder, Disk Utility Program and Simulator) or user-written. It is composed of several separate but closely-related routines; its two principal components are:

- The Skeleton Supervisor, and
- The Monitor Control Record Analyzer

Skeleton Supervisor. This contains the requisite direction and control logic for the orderly transition of one program to another. The Skeleton Supervisor is read into core storage whenever monitor system operation is initially started, and provides the communications link between monitor programs and user programs.

Monitor Control Record Analyzer. This component of the Nonprocess Supervisor reads the monitor control record, prints its contents on the list and/or System Printer, and calls the appropriate monitor program.

Analysis of monitor control records extends over columns 1-5 only, except for the // JOB card. Invalid control records result in an error message and cause an abort. Blank cards are bypassed and not stacker-selected. The card I/O routine, CARDN, in the skeleton is used; if CARDN is not included by the user, the monitor program uses its own card I/O routine. The // JOB control record resets the abort indicator and the effective address for the Nonprocess Working Storage on disk. It can also specify which of logical disk drives 1 and 2 are expected to be operational, and, accordingly, checks the labels on their disk packs when indicated. The // END control record directs the Nonprocess Supervisor into a wait state.

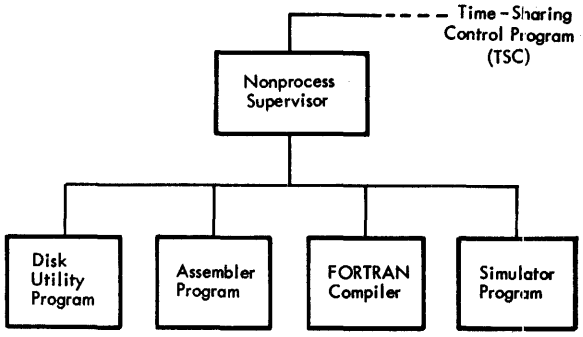


Figure 28. The Nonprocess Monitor

Specifically, the Nonprocess Supervisor performs the following functions:

1. Analyzes all monitor control records (e.g., // JOB, // ASM, // FOR)
2. Performs JOB initialization
3. Calls and transfers control to the requested monitor program (e.g., FORTRAN Compiler, Assembler)
4. Performs PAUS (that is WAIT) and END OF ALL JOB functions when requested
5. Also analyzes control records for the Core Load Builder following the // XEQ, *STORECI and *SIMULCI.

Method of Operation

The Nonprocess Supervisor, including all monitor programs, must reside on logical disk drive zero where it occupies 21 sectors (see System Design Considerations: IBM Nonprocess System). The first 168 words of the Disk Communications Area (DCOM) of sector 00000 on this disk (the system disk pack) contains the Nonprocess Communications Area which provides the logical linkages between monitor programs and user programs. This area holds address words, error indicators (used by DUP, FOR, ASM, SUP, etc.), the name of the program or core load being executed, as well as a loader for the monitor programs.

DCOM is always brought into core each time a // JOB control record is read. Certain words are then initialized to reflect the current status of the disk as reflected by LET/FLET. Note that recognition of a // JOB control record by the Nonprocess

Supervisor also removes all temporary entries from LET. Whenever a // END or // XEQ control card is encountered, DCOM is written back to disk.

Entry to the Nonprocess Supervisor occurs through a) Console Interrupt, b) a CALL SHARE (or CALL VIAQ) statement in a process mainline, c) FORTRAN Compiler, d) Simulator, or e) Disk Utility Program.

In an on-line TSX system, process interrupts are serviced as they occur, the interrupt servicing time being applied against the time specified by the user for nonprocess operations. As an example, assume a process mainline calls for one minute of time-sharing. This one-minute span is the length of time in the share mode. If, during this period, ten seconds are used up for process interrupt servicing, only fifty seconds are actually available for nonprocess work.

If all nonprocess jobs are completed before the end of the user-specified time, the Nonprocess Supervisor program performs a WAIT operation for the remainder of the time allotted. In other words, if the CALL SHARE statement specified one minute of time-sharing, control is not returned to the process program until one minute has elapsed, or alternatively, a CALL ENDTS statement is executed by an interrupt routine (see Use of Time-Sharing).

Figure 29 illustrates, in simplified form, Nonprocess Monitor action during time-sharing.

If a nonprocess job is not completed before the specified time is up, it is saved on the disk. When the next CALL SHARE statement is executed, operation of the nonprocess job is resumed at the point of termination.

When an unfinished job is waiting, the CALL SHARE statement causes it to be read in and executed. Otherwise, the Nonprocess Supervisor program is read into core and determines, by checking a program indicator located within the System Director, if any time-sharing operations are to be performed. This indicator is turned on by the execution of a special console interrupt routine, supplied with the system.

The following example illustrates a typical use of the Nonprocess Monitor whenever nonprocess jobs are ready for execution.

1. Operator stacks jobs in Card Reader and starts Reader.
2. Time-Sharing is typically initiated by an operator interrupt, with a coded number set up in the console switches to indicate a time-sharing request.
3. Interrupt routine sets a program indicator to a process mainline.
4. Process mainline calls for time-sharing when it is idle. It specifies the time interval.
5. Nonprocess programs may be interrupted and later continued by an external (that is, process) interrupt or timer interrupt. This will involve an exchange to the disk save area if the interrupt program is not in core, or if the shared period has timed out.
6. Nonprocess jobs are completed in sequence until no jobs remain (program ends on a WAIT instruction) or until // END OF ALL JOBS control record is reached.
7. During time-sharing, the Skeleton Supervisor will be in transient core, identifying monitor control records and initiating monitor programs.

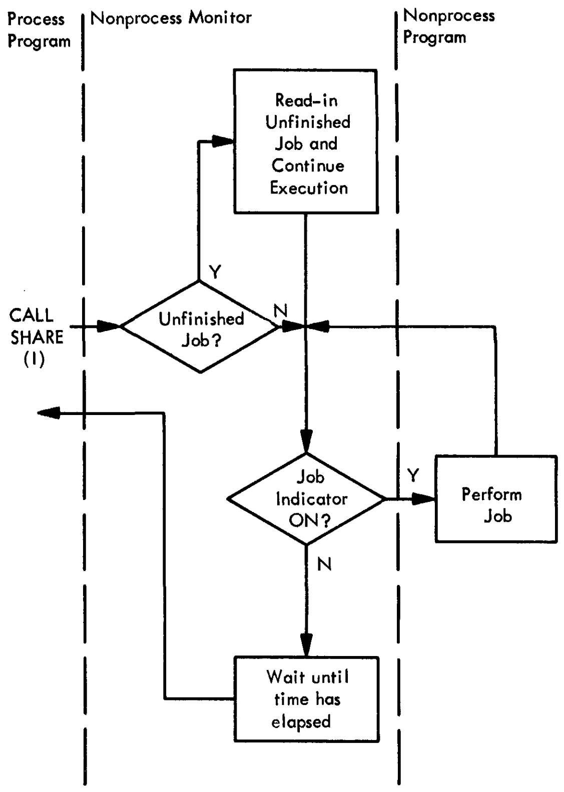


Figure 29. Illustrating Nonprocess Monitor Action during Time-Sharing

DISK UTILITY PROGRAM (DUP)

DUP is a set of routines designed to aid the user in the day-to-day maintenance of data and programs on disk packs. That is, it has the capabilities of storing, deleting, and outputting user's programs as well as defining system and machine parameters. It also updates the location equivalence table (LET) and maintains other communications areas. The Disk Utility Program is called into operation by a // DUP monitor control record; it can be used on-line or off-line.

The // DUP monitor control record must be followed by at least one DUP control statement that selects the desired routine. DUP control statements are identified by an asterisk in column 1. Columns 2 through 10 contain a symbolic code which identifies the routine (e.g. *STORE, *DELETE, *SEQCH). The columns following the coded routine name provide additional information used by the routine itself.

Like the Nonprocess Supervisor, DUP must reside on logical disk drive zero where it occupies 68 sectors. Primary entry to DUP derives from a) Nonprocess Supervisor, b) FORTRAN Compiler, c) Assembler, and d) Core Load Builder.

DUP uses the card I/O routine, CARDN, if this is included in the skeleton; otherwise, it uses its own card I/O routine. Blank cards are skipped and stacker-selected when searching for control records. Non-DUP or non-monitor control records result in an error message. All DUP control records and messages are printed on both the System and List printers.

Essential data for most DUP functions to communicate with a disk pack include the following:

- Disk sector addresses
- Numeric label in word 0, sector 0
- Disk Communications Area (DCOM) -- This provides information on the size and location of work storage areas, LET for the Relocatable Program Area and FLET for the Core Load Area.
- Valid entries in LET/FLET

A list of all DUP functions is given in the Summary of Nonprocess Monitor Control Records. See also Examples of Nonprocess Monitor Usage.

FORTRAN COMPILER

The TSX FORTRAN Compiler is a disk-resident version of the 1800 card compiler, and occupies 103 sectors on logical disk drive zero. Provision is also made for the user to easily make use of input-output, conversion and arithmetic subroutines that are a part of the TSX subroutine library. The FORTRAN language is described in IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715.

The // FOR monitor control record is used to call the FORTRAN compiler into operation, and to name the mainline program. The compiler reads the control records and source program in card form only. After a successful compilation, the object program in relocatable format is moved to the temporary area on disk, and an entry (name and disk block count) is made in LET. It can, henceforth, be called for execution by an // XEQ control record, or it can be stored permanently in the Relocatable Program Area by a DUP (*STORE) operation. All FORTRAN programs are compiled in relocatable format. A list of FORTRAN control records is given in the summary at the end of this section.

ASSEMBLER

The Assembler program for the 1800 TSX system is a disk-resident version of the 1800 card assembler. It is designed to translate source program statements written in a symbolic format into a binary format which may be stored and/or dumped by the Disk Utility Program (DUP), or executed directly from the Nonprocess Work Storage on disk. The Assembler Language is fully described in IBM 1800 Assembler Language, Form C26-5882.

The Assembler program resides on logical drive zero and occupies seven cylinders. Entry to it is obtained via a // ASM monitor control record. The Assembler accepts control records and source programs in card form only. Upon a successful assembly, the object program in relocatable format is moved to the temporary area on disk where it can be called for execution by a // XEQ control record or stored permanently in the Relocatable Program Area by a DUP (*STORE) function. A list of Assembler control records is given in the summary at the end of this section.

SIMULATOR PROGRAM

The simulator is designed as a debugging aid which allows the user to checkout or test process and/or nonprocess programs without disrupting the normal operations of the TSX system -- that is, without taking the system off-line. It functions under the control of the Nonprocess Monitor.

Each instruction in the object program is analyzed for a valid operation code and format before its operation is simulated. In addition, addresses of store and branch instructions are checked to ensure that the instruction would not alter anything outside of the areas of the defined program, COMMON, INSKEL COMMON, or the level work area, if they are actually executed on-line. Process input values may be read from cards or derived from a random number generator. Since System Skeleton routines are used during simulation, it is mandatory that the skeleton area be built before simulation of process core loads can be performed.

Since the primary function of the Simulator is to detect programming errors in the object project, several optional debugging features are available to aid the user. These include Snapshot, Branch Trace and Dump. Simulated COMMON can be dumped on cards so that a run can be executed in several different parts. In addition, the branch and arithmetic trace provided by the FORTRAN Compiler can be operative in the simulator mode.

Simulation runs for process programs are called by a DUP control record, *SIMULCI; runs for nonprocess programs are called by a // SIM monitor control record. Details of operating procedures and stacked-input for a typical simulation run are described in IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Subroutines

General Input/Output

Each time the Simulator encounters a user-called sequence to an I/O subroutine, the location of the calling sequence and the subroutine name are printed on the List printer. Each time the Simulator encounters a subroutine test function (I/O function digit = 0), the following occurs: the first time a test is encountered, a busy return is made; the second time, a not busy return is made. Succeeding entries alternately cause busy and not-busy returns.

Listed below are the general input/output subroutines (IBM-supplied) recognized by the Simulator, and corresponding operations which the Simulator performs:

SUBROUTINE	OPERATION
CARDN (Simulated card subroutine)	Read a card, feed a card, simulate punch a card
DISKN (Simulated disk subroutine)	Read disk, write disk, simulate disk seek
MAGT (Simulated magnetic tape subroutine)	Simulates all read, write, and control functions relative to 2401 and 2402 magnetic tape units
PAPTN (Simulated paper tape subroutine)	Simulate reading paper tape, simulate punching paper tape)
PLOTX (Simulated plotter subroutine)	Simulate plotter output
PRNTN (Simulated printer subroutine)	Print a line, simulate a carriage operation
TYPEN or WRYTN (Simulated printer keyboard subroutine)	Simulates printing on 1816 printer keyboard or 1053 printer

The Simulator requires that the card reader, disk, and List printer be physically present on the system.

Process Input/Output

Call sequences which specify input from pulse input points, digital input points, process contact points, and analog input, may obtain input from two sources: cards and a random number generator.

Data cards are used if samples of specific values are desired; the points can be read in a nonprocess program and punched into cards to be read by the Simulator. Any value can be simulated when using cards, but in order to obtain the desired results, the input data must be sequenced according to the flow of the process input subroutines called. In other words, the card feature requires careful ordering of the card deck.

A random number generator, within the Simulator program, produces numbers that fall into a user-specified range. With this option, the user can employ a wide variety of input data to check program operation. A psuedo-process input environment can also be created through the use of a random number generator. All input values are printed on the list printer as they are called.

In the program being simulated, call sequences that specify output for the contact operate, pulse output, digital output, and digital-to-analog output features are printed when they are encountered. Input call sequences, error messages, and data are included in the printed output. This provides a complete chronological record of all that occurred during the simulation.

IBM-supplied process input/output subroutines are functionally simulated; that is, the subroutines' call parameters are analyzed according to specifications supplied in the form of control records. The routine name, calling parameters, and data are printed on the List printer. Listed below are the process input/output subroutines recognized by the Simulator, and corresponding operations which the simulator performs. Special-condition returns are also simulated.

SUBROUTINE	OPERATION
AIPTN or AIPN (Simulated analog input point)	Simulates the read of a single analog point
AIRN (Simulated analog input random)	Simulates reading random analog input points
AISQN or AISN (Simulated analog input sequential)	Simulates reading sequential analog data points
DAOP (Simulated digital-analog output)	Simulates the transfer of digital or analog information
DICMP (Simulated digital input read compare)	Simulates the reading in of digital input values under program control and compares these values to a table of user-supplied values. Only the first compare error is detected. A single entry to the special routine is made with appropriate indication. The end-of-table interrupt will not occur if a comparator error occurs.
DIEXP (Simulated digital input read expand function)	Simulates the reading in of a digital input value and expands it into 1, 2, 4, 8, or 16 words.
DINP (Simulated digital input hardware functions)	Simulates the reading in of digital input values

Arithmetic and Conversion Subroutines

Copies of the IBM-supplied arithmetic and conversion subroutines are contained within the Simulator. It is these copies that are executed when a call to an arithmetic or conversion subroutine is encountered. The requested operations are performed in a psuedo-processing environment maintained under control of the Simulator.

General TSX Subroutines

When a call to a TSX control subroutine is recognized by the Simulator, the subroutine name and its calling sequence parameters are printed. There are two categories of subroutines designed for control and communication with the TSX system: the termination class and the functional simulate class.

The following subroutines comprise the termination class, and when encountered, cause the Simulator to execute the close-job procedure:

BACK	PAUSE
CHAIN	SPECL
DPART	STOP
INTEX	VIAQ
LINK	EXIT

The subroutines listed below comprise the functional simulate class, and when encountered, cause the Simulator to simulate the function, i.e., they analyze the call parameters for validity and print the routine name, the calling parameters, and the data contained within the subroutine.

CLEAR	REMSK
COUNT	SAVMK
ENDTS	SETCL
LEVEL	SHARE
MASK	TIMER
OPMON	UNMK
QIFON	UNQ
QUEUE	

User-Written Subroutines

User-written subroutines are simulated in the same manner as mainline programs.

Common Area

The simulated COMMON area can be dumped on cards whenever a program being simulated is terminated. The output cards can be used for input to reload COMMON, thus providing communication from one core load to another.

Restrictions

Restrictions placed upon the use of the Simulator program are listed below:

1. Nonprocess work storage must be used if actual data is to be transferred between disk and core.
2. Link or chain jobs must be simulated by presenting one core load at a time.
3. The Simulator utilizes LIBF and CALL instructions for special purposes. When analyzing post-mortem dumps, the contents of LIBF and CALL locations should be ignored by the user.
4. All I/O must be performed by Simulator subroutines. An execute I/O (XIO) instruction is not simulated but will be recorded on the List printer.
5. A wait (WAIT) instruction will be recorded on the list printer.
6. A storage protect setting instruction (STS with both the F-bit and the 9th bit equal to zero) will result in a termination.
7. An attempt to store into a skeleton area other than the INSKEL COMMON and work level areas will result in a termination.
8. Operation codes of 00, 38, 58, 78, and FF are invalid and will result in a termination.
9. A subroutine I/O area parameter pointing to the skeleton will result in a termination.

SUMMARY OF NONPROCESS MONITOR CONTROL CARDS

Tables 5-10 give a brief summary of all Nonprocess Monitor control cards. For details of card preparation and their functions, see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Table 5. Monitor Control Cards

// JOB	Initializes a nonprocess job
// DUP	Reads the disk utility program into core for execution
// XEQ	Reads the user's programs into core for execution
// ASM	Reads the Assembler into core for execution
// FOR	Reads the FORTRAN compiler into core for execution
// SIM	Reads the Simulator program into core allowing a nonprocess program to be simulated
// PAUS	Causes the system to WAIT
// END or	
// END OF ALL JOBS	Signals the Nonprocess Supervisor that all nonprocess work is complete

Table 6. Loader Control Cards

*INCLD	Causes a named program to be included in the skeleton or in a mainline core load
*RCORD	Records interrupts that occur during the execution of process core loads
*FILES	Provides for the designation of disk areas to be used by the FORTRAN program in which the files were defined
*CCEND	Indicates that no loader control cards are following and the core load builder can be called
*LOCAL	Permits groups or blocks of subprograms to be loaded into core when they are called

Table 7. DUP Control Cards

*DEFINE OCORE	Specify the size of object core
*DEFINE NDISK	Specify the number of disk drives on the system
*DEFINE CONFG	Specify the system configuration with respect to disk areas
*DEFINE REMOV	Allow the user to delete FORTRAN or the Assembler from the monitor disk
*DEFINE PAKDK	Pack relocatable programs into unused areas identified by *DELET
*DLABL	Labels a disk pack and, if not system pack, writes addresses
*STORE	Stores relocatable programs in the Relocatable Program Area (user or temporary) on disk
*STOREDATA	Stores blocks of data in Core Load (core image) Area on disk
*STORECI	Causes a core load to be built and stored in the Core Load Area on disk
*DUMP	Dumps programs from the disk to the system I/O device or list printer
*DUMPDATA	Dumps blocks of data as indicated in *DUMP
*DUMPLET	Dumps LET and/or FLET on the list printer
*DELET	Replaces a program name in LET or FLET with the name 9DUMY thus making the program area available to the store function
*DWRAD	Allows the user to write addresses on a specified area of disk
*STOREMD	Allows the user to modify existing nonprocess core loads and relocatable programs without previously deleting them
*SEQCH	Used to change the sequence of existing core load linkages for process or nonprocess core loads
*SIMULCI	Reads the Simulator program into core, allowing a process program to be simulated
*DICLE	Allows the user to modify the interrupt core load table

Table 8. FORTRAN Control Cards

* IOCS (CARD, TYPEWRITER, KEYBOARD, 1443 PRINTER, PAPER TAPE, MAGNETIC TAPE, DISK, PLOTTER)	Delete any not used
** Header information to be printed on each compiler output page	
* ONE WORD INTEGERS	(Store integer variables in one word) This function is automatic in process programs.
* EXTENDED PRECISION	(Store floating point variables and constants in 3 words instead of 2)
* ARITHMETIC TRACE	(Switch 15 ON to print result of each assignment statement)
* TRANSFER TRACE	(Switch 15 ON to print value of IF or Computed GO TO)
* LIST SOURCE PROGRAM	(List source program as it is read in)
* LIST SUBPROGRAM NAMES	(List subprograms called directly by compiled program)
* LIST SYMBOL TABLE	(List symbols, statement numbers, constants)
* LIST ALL	(List source program, subprogram names, symbol table)
* NONPROCESS PROGRAM	(Identifies this compilation as a nonprocess program)
* PUNCH	(Causes DUP to punch an object deck after successful compilation)

Table 9. Assembler Control Cards

*TWO PASS MODE	Read source deck twice; must be specified when *LIST DECK or *LIST DECK E is specified, or when intermediate output fills working storage
*LIST	Print a listing on the principal printing device
*LIST DECK	Punch a list deck on the principal I/O device (requires *TWO PASS MODE)
*LIST DECK E	Punch only error codes (cc 18-19) into source program list deck (requires *TWO PASS MODE)
*PRINT SYMBOL TABLE	Print a listing of the symbol table on the principal printing device
*PUNCH SYMBOL TABLE	Punch a list deck of the symbol table on the principal I/O device
*SAVE SYMBOL TABLE	Save symbol table on disk as a system symbol table
*SYSTEM SYMBOL TABLE	Use system symbol table to initialize symbol table for this assembly
*PUNCH	A relocatable binary deck will be punched by DUP following this assembly
*OVERFLOW SECTORS n	n = number of sectors of nonprocess working storage allowed for symbol table overflow
*COMMON n	n = length of COMMON in words (decimal)

Table 10. Simulator Control Cards

*SNAP	Displays up to 10 locations following execution of an instruction
*TRACE	Traces or displays same information as for *SNAP
*DUMP	Dumps simulated core storage
*SAVE COMMON	Punches out binary deck of process and variable COMMON
*LOAD COMMON	DEFINES and analyzes COMMON from *SAVE COMMON OUTPUT deck
*XIO	Suppresses printing of IOCC words referenced by XIO instruction
*WAIT	Suppresses printing of WAIT instructions
*START SIMULATION	Signals that all Simulator control cards have been read
*END DATA	Terminates Simulator run

EXAMPLES OF NONPROCESS MONITOR USAGE

The prime purpose of this section is to illustrate a few of the many possible uses of 1800 TSX features, and to accentuate the many more possibilities based upon the ability of the user to apply the basic concepts and techniques. Numerous sample programs and coding examples are presented as demonstration of good programming practice and technique. These examples conform strictly to standard TSX coding conventions.

The JOB

When a programmer is given a problem, he analyzes that problem and defines a precise problem-solving procedure: that is, he writes a program or a series of programs. To the monitor system, executing a mainline program (and any subroutines and subprograms that it calls) is a job step. A job consists of executing one or more job steps.

At its simplest, a job consists of one solitary job step. For example, assembling or compiling a program is a job consisting of one job step. Similarly, executing a FORTRAN mainline program to invert a matrix is a job consisting of a single job step.

If the problem is complex, one job may consist of a series of job steps. Such a job may include multiple assemblies, compilations, disk utility functions, and executions. A job always begins with a // JOB control card which is the first statement in the sequence of control statements that describes a job.

The JOB Deck

The input to the Nonprocess Monitor may consist of one or more job decks. Each job deck is preceded by a // JOB. The processing of each job deck is controlled by the Supervisor program as specified in the monitor control cards. As an example, consider the following stacked input arrangement (see Figure 30).

The above sequences will compile, store and execute both program PROG1 and program PROG2 provided that:

1. There are no source program errors, and
2. There is sufficient room in the Nonprocess Work Storage area.

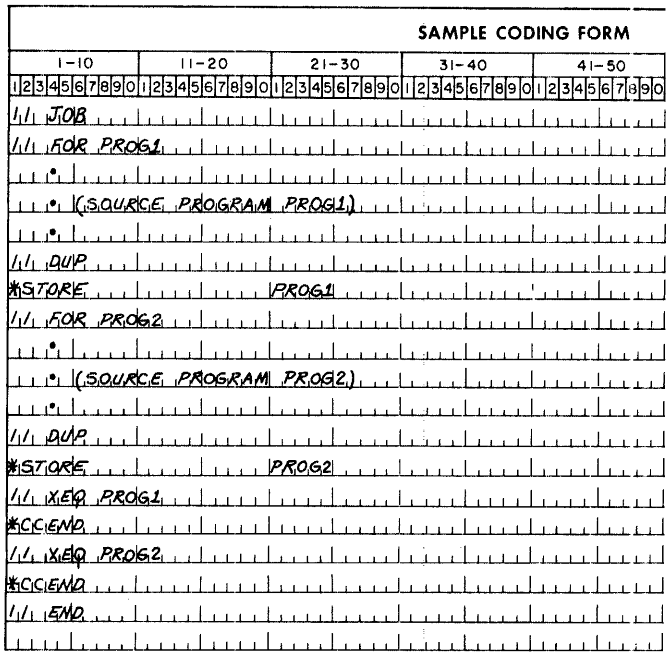
A source program error will cause the DUP Store Operation to be bypassed for that program, and all following // XEQ requests preceding the next // JOB card will be disregarded. This feature (that is // XEQ -- request disregard) can prove very useful when the successful execution of one program depends upon the successful completion of the previous program. A combination such as this should be considered as one job. The // XEQ control cards should not be separated by a // JOB card. Note from Figure 30 that it would not be necessary to store the two programs if they were executed on a one-shot basis.

Assembling/Compiling Programs

Programs are of two types: process and nonprocess. A process program is one that continuously monitors a control process. All application programs are, by definition, process programs. A nonprocess program, on the other hand, is not directly related to the control process itself. An assembler program is an example of a nonprocess program: other examples include compilers, data reduction, payroll, bookkeeping, simulation of new and existing programs, and linear programming.

Process and nonprocess programs may be further classified as main programs or subroutines. Subroutines can be subdivided into the following: LIBF (library functions), CALL, Interrupt, IBM-supplied, and LOCAL subroutines.

In the off-line or time-sharing mode of operation, the user may exercise any of four options in assembling/compiling and executing a nonprocess program. Figure 31 illustrates these approaches in simplified form. A distinction should be drawn between process and nonprocess programs. The initial process program can only be executed through a cold start procedure for an on-line TSX system. If the process, mainline, or combination core load is already disk-resident (in the Core Load Area) it is called by a CALL CHAIN or CALL QUEUE.


SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
// JOB
// FOR PROG1
   *
   * (SOURCE PROGRAM PROG1)
   *
// DUP
*STORE              PROG1
// FOR PROG2
   *
   * (SOURCE PROGRAM PROG2)
   *
// DUP
*STORE              PROG2
// XEQ PROG1
*CCEND
// XEQ PROG2
*CCEND
// END
```

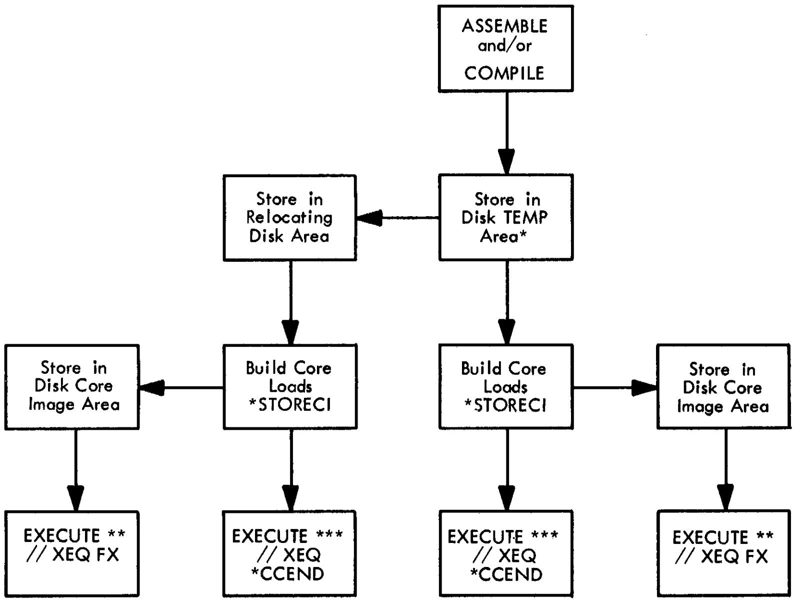
Figure 30. Illustrating a JOB

EXAMPLE 1. ASSEMBLE AND EXECUTE A NONPROCESS PROGRAM FROM THE TEMPORARY AREA (see Figure 32).

The Assembler is unable to differentiate between process and nonprocess programs -- these are treated alike. Following assembly, the object program in relocatable format is moved to the temporary area on disk, and its entry (name, word count, and sector address) made in LET.

If the user desires to perform only an initial check on his program, and not execute it, // XEQ and *CCEND are not required. If he plans to verify the program logic and results (if any), he will execute it. The presence of the // XEQ and *CCEND control cards calls in the Core Load Builder, and a core load is built and executed. In addition, a listing of source statements as well as the corresponding object program, and a directory of all valid labels used in the program can be obtained by specifying these options with the appropriate Assembler control cards. The order in which programs are assembled is important when the *SAVE SYMBOL TABLE control card is used in assembling related programs.

Note that the relocatable program will reside in the temporary disk area until it is deleted by the next // JOB card. An *CCEND control card must always follow an // XEQ card if a relocatable program is referenced in the // XEQ card.


ASSEMBLE and/or COMPILE
Store in Relocating Disk Area
Store in Disk TEMP Area*
Store in Disk Core Image Area
Build Core Loads *STORECI
Build Core Loads *STORECI
Store in Disk Core Image Area
EXECUTE ** // XEQ FX
EXECUTE *** // XEQ *CCEND
EXECUTE *** // XEQ *CCEND
EXECUTE ** // XEQ FX

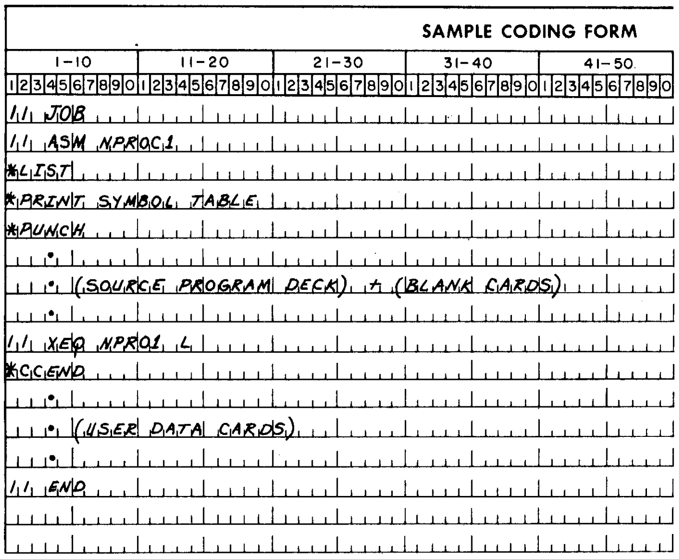
Note:

* This is automatic if the assembly or compilation is successful.

** Execution occurs through a Cold Start, CALL LINK or // XEQ.

*** Execution can only occur through a // XEQ.

Figure 31. Assemble/Compile and Execute a Nonprocess Core Load



SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
1234567890123456789012345678901234567890123456789 0
// JOB
// ASM NPROC1
*LIST
*PRINT SYMBOL TABLE
*PUNCH
   .
   . (SOURCE PROGRAM DECK) + (BLANK CARDS)
   .
// XEQ NPRO1 L
*CCEND
   .
   . (USER DATA CARDS)
   .
// END
```

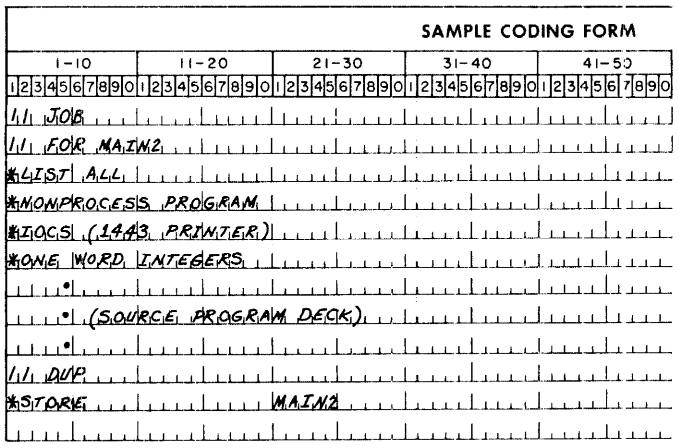
Figure 32. Assemble and Execute a Nonprocess Program from the Temporary Area

EXAMPLE 2. COMPILE AND STORE A NONPROCESS PROGRAM IN THE RELOCATABLE PROGRAM (OR USER) AREA ON DISK (see Figure 33).

Unlike the Assembler, the Fortran Compiler distinguishes between the two types of programs by the absence or presence of the *NONPROCESS PROGRAM control card. In a process program, each integer variable automatically occupies one word of storage. In a nonprocess program, however, the *ONE WORD INTEGERS control card forces the compiler to allocate one word of storage to each integer variable; in the absence of this card, the same allocation (that is, two words) for real variables is made. In the case of a large array, this could be prohibitive.

All FORTRAN programs are compiled in relocatable format. Following compilation, the relocatable object program is moved to the temporary disk area, and an entry made in LET. It can now be called for execution or loaded to the Relocatable Program (that is, the User) Area on disk.

In Example 2, the relocatable program MAIN2 is stored in the Relocatable Program (User) Area. The actual storing of the program consists of physically moving the program to its destination area (the User Area) from the temporary area of Nonprocess Working



SAMPLE CODING FORM

```
// JOB
// FOR MAIN2
*LIST ALL
*NONPROCESS PROGRAM
*IOCS (1443 PRINTER)
*ONE WORD INTEGERS
    •
    • (SOURCE PROGRAM DECK)
    •
// DUP
*STORE              MAIN2
```

Figure 33. Compile and Store a Nonprocess Program in the Relocatable Program Area (User Area) on Disk

Storage. When the storing operation is completed, LET is updated and the communications and fixed area parameters reset to reflect these changes.

Note that a store from the temporary (TEMP) area to the permanent Relocatable Program Area causes TEMP to be packed to reflect that program moved. An exception exists when the program is the last entry in TEMP or when there is only one TEMP program initially.

EXAMPLE 3. COMPILE AND EXECUTE A NONPROCESS PROGRAM FROM THE CORE LOAD AREA (see Figure 34).

This example illustrates the third and fourth options which may be taken to assemble/compile and execute a nonprocess program from the Core Load Area. Note that subroutines TIMSB, ERROR, and PRINT are compiled and stored in the User Area as these subroutines are frequently referenced by this and other nonprocess programs. In building process mainline and combination core loads, it may not be necessary to store these subroutines. The store core image routine is used to store a program in core image form (as a core load) in the core load area and to assign the core load a name. By making column 9 of the *STORECI control card non-blank, a map of the locations and names of subroutines and subprograms loaded with the core load is obtained. When the nonprocess core load is correctly built, DUP will search through its program name table, find the name of the core load just built, and add its disk address and word count to the table. In addition, any programs referenced in this core load name table are looked up in FLET and their disk addresses and word counts added to the table. The core load is then executed from the Core Load Area. FX in columns 16 and 17 of the // XEQ monitor control card signifies that the input program is in core image format and that FLET is to be searched for this program name.

Deleting and Replacing Relocatable Programs, Core Loads, and Data Files

The *DELETE function allows the user to delete any named

Relocatable program
Mainline core load
Combination core load
Interrupt core load
Nonprocess (or link) core load
Data file

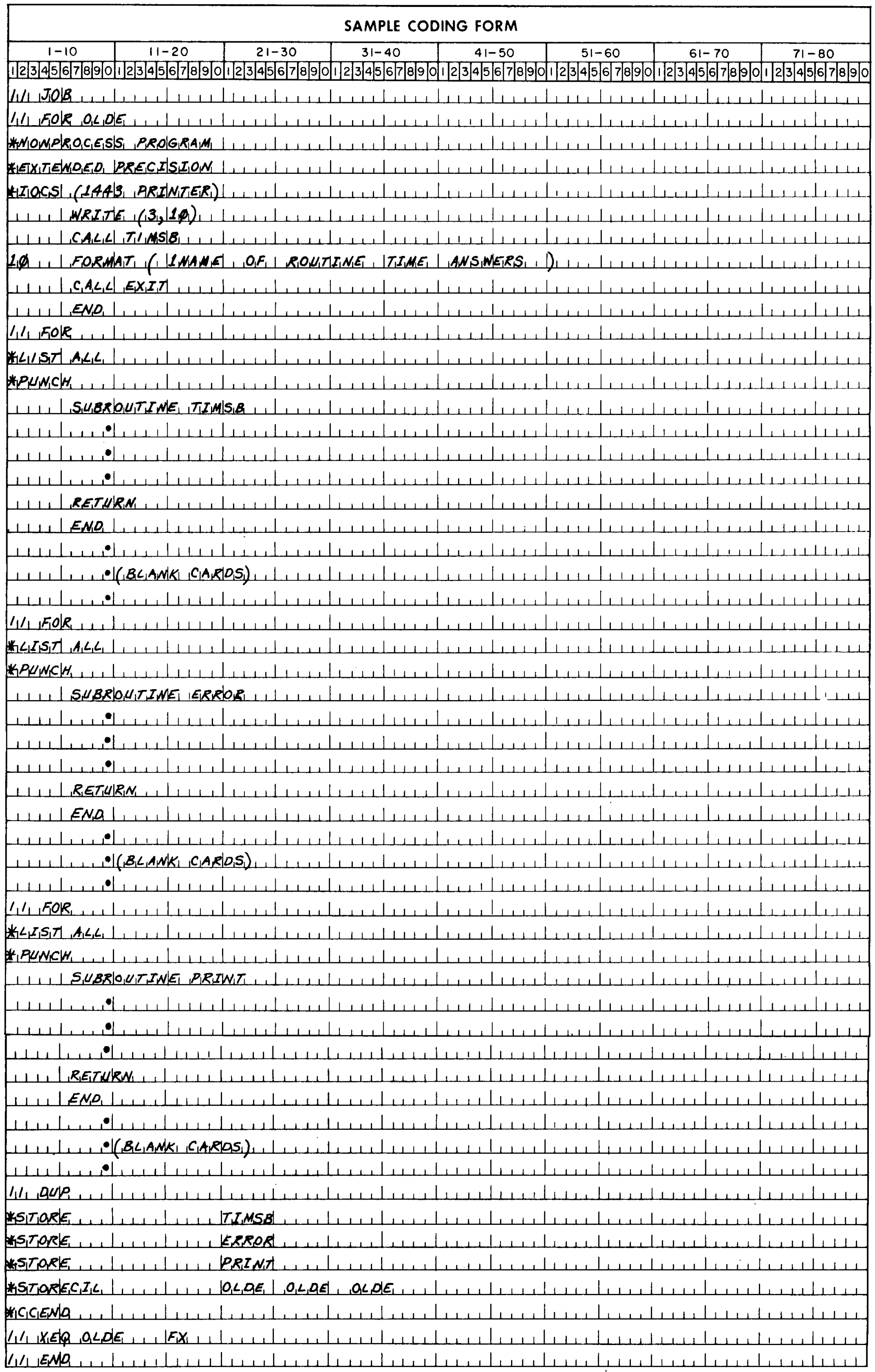
from the disk. An entry of a program in LET or a core load/data file in FLET takes the normal form

LET:	NAME	DISK BLOCK COUNT

FLET:	NAME	WORD COUNT	SECTOR ADDRESS

where each LET and FLET entry occupies three and four words of disk space respectively. Whenever a program or a core load is deleted, its NAME in LET or FLET is replaced by the symbolic 9DUMY and henceforth the system is no longer cognizant of this program or core load. Furthermore, the area on disk previously occupied by a program or core load is now available for the storage of other programs, core loads, or data files. These areas are available, but only used after all previously available areas have been used.

A core load may be deleted and, in addition, have its reference replaced by another core load's word count and sector address. The replacement core load must be of the same type. That is, a mainline



SAMPLE CODING FORM

```
// JOB
// FOR OLDE
*NONPROCESS PROGRAM
*EXTENDED PRECISION
*IOCS (1443 PRINTER)
      WRITE (3,1Ø)
      CALL TIMSB
1Ø    FORMAT (1NAME  OF  ROUTINE  TIME   ANSWERS  )
      CALL EXIT
      END
// FOR
*LIST ALL
*PUNCH
      SUBROUTINE TIMSB
         •
         •
         •
      RETURN
      END
         •
         •(BLANK CARDS)
         •
// FOR
*LIST ALL
*PUNCH
      SUBROUTINE ERROR
         •
         •
         •
      RETURN
      END
         •
         •(BLANK CARDS)
         •
// FOR
*LIST ALL
*PUNCH
      SUBROUTINE PRINT
         •
         •
         •
      RETURN
      END
         •
         •(BLANK CARDS)
         •
// DUP
*STORE              TIMSB
*STORE              ERROR
*STORE              PRINT
*STORECIL           OLDE  OLDE  OLDE
*CCEND
// XEQ OLDE    FX
// END
```

Figure 34. Compile and Execute a Nonprocess Program from the Core Load Area

core load may be replaced by another mainline core load, an interrupt core load by an interrupt replacement core load, a combination core load by a replacement core load, and a nonprocess core load by a nonprocess replacement core load. Replacement of the four types of core loads is governed by certain rules which are summarized as follows. Note that the replacement function within an *DELETE operation does not alter the core load name, but only its word count and sector address.

Combination and Interrupt Core Loads. In deleting a combination or interrupt core load, all references to this core load in the Program Name Table (PNT) and/or Interrupt Core Load Table (ICLT) must be replaced by a replacement core load name. Absence of this specification in the control card invalidates the deleting function. Furthermore, if an interrupt core load or combination is used to service multiple interrupts, all interrupt core load entries in the PN Tables, Queue Table, and ICL Table are automatically replaced with a single delete operation by specifying 9999 for the interrupt level and bit positions on the control card (columns 39-42).

The rule is never to allow a previous serviceable level and its bit indicator to remain unserviceable.

Mainline and Nonprocess (or Link) Core Loads. In general, a mainline or nonprocess core load that is not currently being called by other core loads does not require replacement. If, however, it is still being referenced in the Queue Table, the PNT within the System Skeleton or some other PNT, deletion is restricted because it is still necessary to maintain this core load identity in the system. Note that a nonprocess core load may be deleted without a replacement core load even though it is still referenced. A negative value is then placed in the word count position of the PNT entry in those core loads referencing the deleted nonprocess core load. A nonprocess core load is also referred to as a link.

Data Files. By definition, a data file is an area in the Core Load Area established by an *STOREDATA function with a D in column 11. Data files can be deleted but not replaced. In deleting a data file from the disk, the user should be aware that the system does not check to see whether this data file is still being referenced by currently executing core loads. This means that if he wishes to delete a data file, he has to ensure by some programming means that there is no reference to this file: that is, no reading from or writing to this file. If there is a reference, there is a distinct possibility that core loads writing to or reading from this file might destroy one or more core loads stored in the same location the data file was located.

EXAMPLE 4. DELETE A PROCESS MAINLINE, COMBINATION OR INTERRUPT CORE LOAD FROM THE CORE LOAD AREA (see Figure 35).

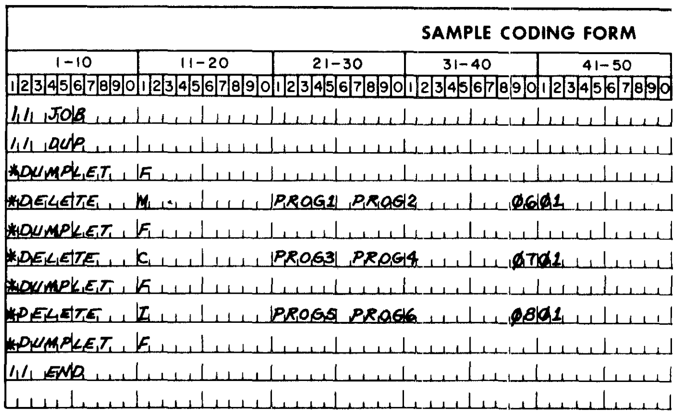
In deleting a process mainline core load, the user should ensure that this core load is not being referenced or called by any other core load that may in turn reference further core loads. If such a situation exists, up to 14 names of calling core loads will be listed; if the number of calling core loads exceeds 14, any excess will not be indicated in the error message. The solution here is to eradicate the excess core loads from the Fixed Area, either by a sequence change or a deletion.

The delete operation is merely one of removing or eliminating an entry from the FLET table with a system mnemonic name 9DUMY, indicating an unused area on disk. Note that in a fresh (that is, new) disk pack, the Core Load or Core Image Area is initially represented in FLET by a 9DUMY entry thus:

NAME (= 9DUMY)
SECTOR COUNT
SECTOR Address

Subsequent *STORECI operations will move this entry. A delete simply replaces a core load with a 9DUMY. In practice, a delete is normally followed by a replacement unless the core load being deleted is considered "dead," thus making its replacement unnecessary.

Example 4 also demonstrates the use of *DUMPLET as an effective programming tool. A dumplet following a delete operation is good programming practice; it shows conclusively that a program



SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
1234567890123456789012345678901234567890123456789O
// JOB
// DUP
*DUMPLET  F
*DELETE   M         PROG1 PROG2       Ø6Ø1
*DUMPLET  F
*DELETE   C         PROG3 PROG4       Ø7Ø1
*DUMPLET  F
*DELETE   I         PROG5 PROG6       Ø8Ø1
*DUMPLET  F
// END
```

Figure 35. Delete a Process Mainline, Combination, or Interrupt Core Load from the Core Load Area

or core load is in fact removed from the FLET table. For an understanding of LET/FLET tables, the user is referred to the Systems Reference Library: IBM 1800 Time-Sharing Executive System, Operating Procedures, Appendix F, Form C26-3764.

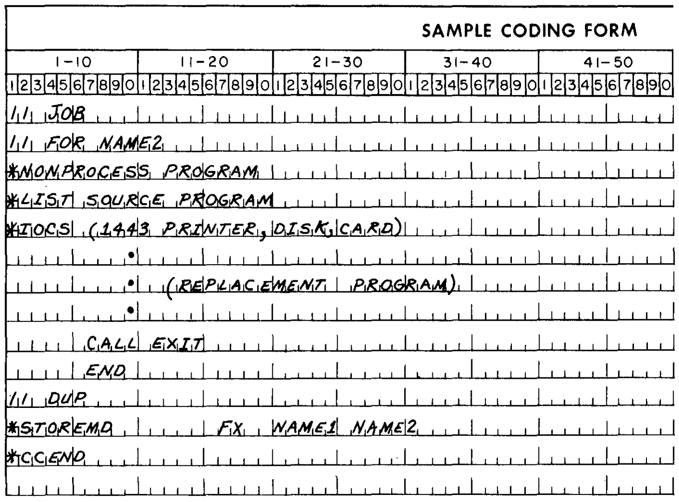
In all three cases, the FLET table is searched for the core load name to be deleted, and its replacement name. Any references to the old program in the Program Name Table of all core loads are then replaced with the word count and sector address of the replacement core load. The old program name is finally deleted from the FLET table.

In the case of combination and interrupt core loads, the interrupt level and PISW bit position indicators are obtained from the card buffer, converted, and stored in the nonprocess communications area. The ICL Table is then updated.

Note also that in all cases, except for the deletion or replacement of nonprocess programs, a check of the queue in the skeleton is made to see if the program to be deleted or replaced is in the queue. If it is, the queue is updated.

EXAMPLE 5. REPLACING A NONPROCESS CORE LOAD IN THE CORE LOAD AREA (see Figure 36). Like process mainline, combination and interrupt core loads, a nonprocess core load can also be deleted and replaced by an *DELETE operation (see Figure 35).

A nonprocess core load can also be replaced by storing a replacement core load to the Fixed Area, as illustrated in Figure 36. The user can thus modify existing nonprocess core loads without previously deleting them.


SAMPLE CODING FORM

```
// JOB
// FOR NAME2
*NONPROCESS PROGRAM
*LIST SOURCE PROGRAM
*IOCS (1443 PRINTER, DISK, CARD)
         .
         .  (REPLACEMENT PROGRAM)
         .
      CALL EXIT
      END
// DUP
*STOREMD        FX  NAME1 NAME2
*CCEND
```

Figure 36. Replacing a Nonprocess Core Load in the Core Load Area

This is achieved by an *STOREMD operation. An *STOREMD with a Fixed Area destination is exactly equivalent to an *STORECI of a nonprocess core load provided that

1. The replaced entry must be in FLET for a Core Load Area
2. If the function is to modify the Core Load Area, the existing FLET entry must be for a nonprocess core load.

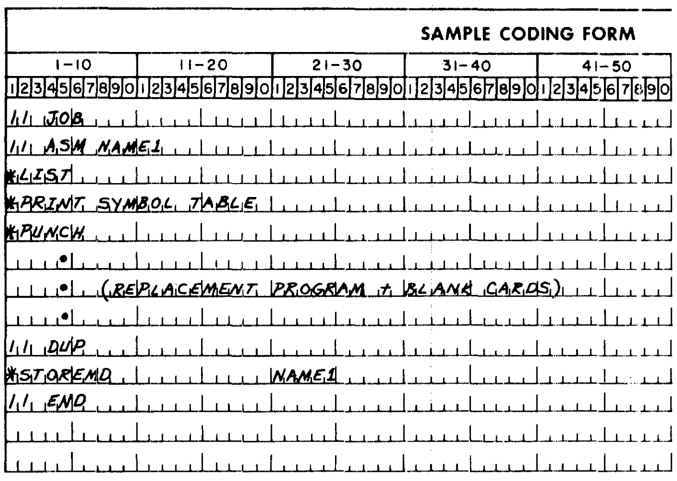
A search through FLET is first made to see if the replacement core load name is already an entry. A further search is then made for a large enough 9DUMY entry to contain the core load. On a find, the sector count of the 9DUMY is checked against the required sector count. The check is successfully terminated by locating a large enough entry on a specified drive which can also take an additional FLET entry. A successful find supplies a destination sector address, and, if previously unknown, the logical drive. Once it is determined that there is space to store the core load, the core load Program Name Table is updated.

Note that the replacement program can either be in the temporary area (of Nonprocess Working Storage) or in the Relocatable Program (that is, User) Area on disk. The name assigned to this program must not be the same as that of the program to be replaced. In Example 5, NAME1 and NAME2 designate two different names. NAME1 (which was previously resident in the Core Load Area) is deleted from the Fixed Area and its entry in FLET removed. The replacement core load NAME2 is stored in the Core Load Area and its name, size in words, and starting sector address then entered into the FLET table.

EXAMPLE 6. REPLACE A RELOCATABLE PROGRAM IN THE USER AREA (RELOCATABLE PROGRAM AREA) (see Figure 37).

Figure 37 illustrates a sequence of control cards that might be used to accomplish this. NAME1 is the name of the replacement program being stored. It must be compiled or assembled with the identical name of the relocatable program being replaced (that is, also NAME1), and it must be the prime entry point. This name must be in the temporary area of Nonprocess Working Storage.

Note that the control card name for the existing program to be replaced must have a LET entry of the same name for a User Area replacement. The replacement program will not overlay the current program, but only cause it to be deleted from the LET table. Thus, the size of a replacement program and the number of entry points in a relocatable-



SAMPLE CODING FORM

```
// JOB
// ASM NAME1
*LIST
*PRINT SYMBOL TABLE
*PUNCH
    •
    •  (REPLACEMENT PROGRAM + BLANK CARDS)
    •
// DUP
*STOREMD            NAME1
// END
```

Figure 37. Replace a Relocatable Program in the Relocatable Program Area

program are governed only by the standard restrictions on any *STORE operation, and not by the size and number of entry points of the existing program.

Other than the above considerations, an *STOREMD with a User Area destination is essentially equivalent to an ordinary *STORE function to the User Area: the same coding is thus used for storing the program. This procedure is mainly used for the modification of existing user-written or IBM-supplied programs.

Changing Core Load Linkages. Through the *SEQCH function, the user is given a powerful programming tool to alter the sequence or order of existing core load linkages for either process or nonprocess core loads. This means that he can now modify a core load Program Name Table such that all references to a core load originally specified will subsequently reference a replacement core load. Note that no deletion of core loads takes place as in an *DELETE with-replacement operation.

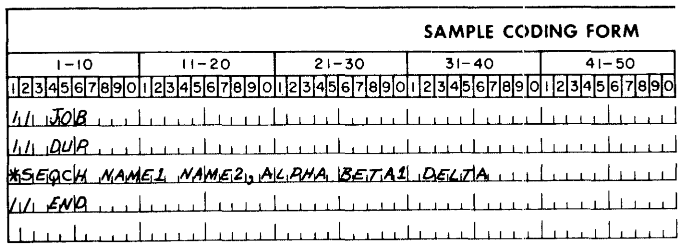
This is known as selective replacement, since the existing referenced core load, the replacement core load, and all other core loads in which changes are to be implemented are all specified.

Note also that the replacement and existing core loads must be type-compatible. That is, a mainline or combination core load may replace either a mainline or combination core load, but a nonprocess or link core load may only be replaced and called by a link. Process calls may emanate from any type of core load.

EXAMPLE 7. CONSIDER THE FOLLOWING SITUATION. In a typical operation, core loads ALPHA, BETA1, and DELTA will call or reference core load NAME1 by a CALL QUEUE statement, thus:

CALL QUEUE (NAME1, 1, 0)

The user now elects to replace NAME1 by NAME2 such that all further references to NAME1 by ALPHA, BETA1, and DELTA will be changed to NAME2. NAME1 can either be a combination or mainline core load resident in the Core Load Area; by definition, NAME2 must either be a combination or mainline core load -- assume that it is also stored (by an *STORECI) in the Core Load Area. The following sequence of control cards may be used to effect this change.



SAMPLE CODING FORM

```
// JOB
// DUP
*SEQCH NAME1 NAME2, ALPHA BETA1 DELTA
// END
```

This will modify the Program Name Table of each of the core loads ALPHA, BETA1, and DELTA so that whenever they call NAME1, the call will refer to NAME2.

At this point in the operation, the user may have no further use for this sequence change, and may well delete core load NAME1, thus:

```
*DELETE          M          NAME1
```

In practice, however, he will probably not delete NAME1 but prefer to return to his original sequence, thus:

```
*SEQCH NAME2 NAME1, ALPHA, BETA1, DELTA
```

Note that because of the type-compatibility between existing and replacement core loads (mentioned earlier), a restriction exists in the case of nonprocess core loads. If, for example, NAME1 were a nonprocess core load, then NAME2, ALPHA, BETA1, and DELTA must, by definition, be also nonprocess core loads.

EXAMPLE 7A. AN ALTERNATIVE METHOD (TO THE *STOREMD FUNCTION) OF ON-LINE RE-

BUILDING OF PROCESS CORE LOADS. Figure 38 illustrates the technique employed, where

- CLA1 is the core load name to be modified; assume the core load is on disk. RELPR is the relocatable program which has been modified.

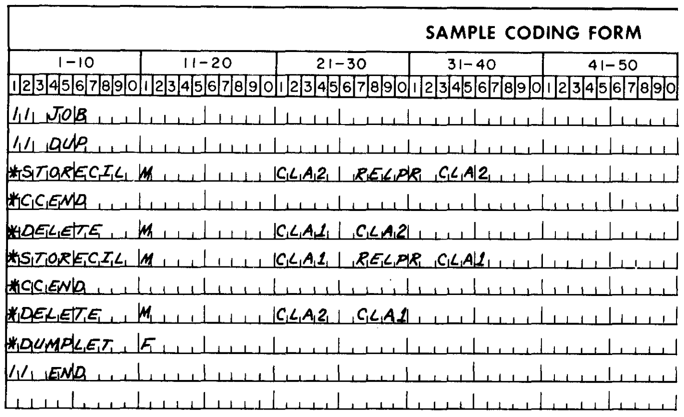
- CLA2 is a temporary core load name used to achieve proper deletion and replacement of the new version of CLA1.

Debugging Core Loads using the Simulator

Several options are available to the user for the debugging of process and nonprocess programs. These are summarized below:

Nonprocess Programs

1. Using TASK (with TASK EAC) in an off-line system only.
2. Using the Simulator in an (a) off-line, or (b) on-line system.


SAMPLE CODING FORM

```
         1-10        11-20       21-30       31-40       41-50
1234567890  1234567890  1234567890  1234567890  1234567890
// JOB
// DUP
*STORECIL M         CLA2  RELPR CLA2
*CCEND
*DELETE   M         CLA1  CLA2
*STORECIL M         CLA1  RELPR CLA1
*CCEND
*DELETE   M         CLA2  CLA1
*DUMPLET  F
// END
```

Figure 38. On-Line Rebuilding of Process Core Loads

Process Programs

1. Using TASK (with TASK EAC) in an off-line system only. To do this, the process program must first be written as a nonprocess program; when fully tested, it is reconstituted into a process program for execution (in an on-line environment).
2. Using the Simulator in either the on-line or off-line environment.

The advantages of the Simulator as a debugging tool lie mainly in

1. The powerful diagnostic messages printed by the Simulator, which allow the user to determine the logic flow of the program by noting the subroutines called, and,
2. in the fact that a process or nonprocess core load may be fully tested without taking the system off-line.

The following examples illustrate the simulation of assembly language process programs.

EXAMPLE 8 (PROGRAM LISTING NO. 2). This program is written for the purpose of debugging the Simulator. If the Simulator erred in the reading of analog input cards, error messages would be printed.

Actual simulation is initiated after the core load build function has been completed. The first thing done by the Simulator is to read the Simulator control cards *XIO and *START SIMULATION. Note that any other Simulator control cards, such as *DUMP, *SNAP, etc., should precede the *START SIMULATION card: data cards should follow the *START SIMULATION card. Since no control card is used to describe the source of analog data, it is assumed that this data will emanate from card input.

After reading the control cards, the Simulator will proceed to interpret the instructions in the user's program, exactly as in execution. The first instruction being a LIBF AISQN, the Simulator prints the S50 message, giving the name of the routine and the absolute address of the LIBF. The S20 message is printed by the Simulator AISQN routine and consists of a description of the calling sequence. Since it is an analog input, a data card is read. However, due to the fact that column 5 is blank, the Simulator is not aware of the format of data on this card and informs the user accordingly with the S12 message. This is likewise the case with the next two cards. Note that the Simulator is still in the process of simulating the first AISQN call. It will continue reading cards until it completes this call. The next card read has a D in column 5, implying digital data input.

However, an absence of the E parameter in column 72 signifies an end of data and thus the S15 message. Only the number +00123 is read into the buffer since the word count is 2, one word of which is the analog address.

Upon completion of the first LIBF, the busy test is encountered. The Simulator will always take the busy exit the first time through a busy test. The second time through, it will exit at the not-busy instruction. Thus, if the busy exit contains an MDX back to the busy test, the Simulator output will show two "goes" through the busy test, one after another, as in the printout of the two S21 messages.

Next, in sequence, another LIBF AISQN is encountered. Again, the S50 message identifies the subroutine and the absolute address of the LIBF. The S20 message gives the calling sequence. A card is read with correct format and an E in column 72. However, only the numbers 1234 and FF12 are read, the first blank terminating the data. Since the word count for AREA2 is four, there is insufficient data on the cards to fill the buffer. Hence the S16 message.

Note that an E in column 72 terminates the call to the subroutine. Therefore, if there had been no E in the last card read, the Simulator would have tried to fill the buffer with data from the next card. The busy test following this is then simulated.

The last call recognized is that to VIAQ, and this terminates the job. The following S99 message is a snapshot of the instruction which caused termination.

Anytime a job is terminated, a snapshot is given to allow the user to determine why the job was terminated.

If the user had wished to see the status of registers at some point of the program, a *SNAP or a *TRACE card could have been added giving the relative address (obtained from the assembly) of the instruction. Note also that the WAIT instruction can be used as a trace aid since the Simulator automatically gives a snapshot of registers upon encountering a WAIT.

PROGRAM LISTING NO. 2 -- EXAMPLE 8

```
// JOB
// *        TEST CASE DB638 START
// ASM DB638
  *LIST


                        *THIS TEST CHECKS ABILITY OF THE SIMULATOR TO
                        *        1. REJECT DATA CARDS WITH INCORRECT FORMAT
                        *        2. SKIP EXCESS DATA CARDS
                        *        3. SENSE INSUFFICIENT DATA IS SUPPLIED
                        *
                        *PRINTER MESSAGES UPON SUCCESSFULL TEST SHOULD BE
                        *    S12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
                        *    S15 TOO MUCH DATA, READ CARDS UNTIL E IN COL 72
                        *    S16 INSUFFICIENT DATA TO FILL I/O AREA, JOB
                        *    CONTINUED
  0000 20 01262615      START LIBF    AISQN
  0001 0  1000                DC      /1000
  0002 1  0020                DC      AREA1
  0003 0  3000                WAIT                    NOT EXECUTED
  0004 20 01262615      NEXT  LIBF    AISQN
  0005 0  0000                DC      /0000
  0006 0  70FD                MDX     NEXT
  0007 0  C01B                LD      AREA1+3         CHECK THAT NO MORE
  0008 01 4C200017            BSC  L  ERR1,Z          THAN 1 DATA CARD WAS
  000A 20 01262615      NEXT1 LIBF    AISQN           USED FOR DATA.
  000B 0  1000                DC      /1000
  000C 1  0024                DC      AREA2
  000D 0  3000                WAIT                    NOT EXECUTED
  000E 20 01262615      NEXT2 LIBF    AISQN
  000F 0  0000                DC      /0000
  0010 0  70FD                MDX     NEXT2
  0011 0  C016                LD      AREA2+4         CHECK THAT I/O AREA
  0012 0  9073                S       CONST           ABOVE DATA CARDS WAS
  0013 01 4C20001B            BSC  L  ERR2,Z          NOT ZEROED.
  0015 30 25241600            CALL    VIAQ
  0017 30 14162897      ERR1  CALL    MESSP
  0019 1  0029                DC      ERR1P
  001A 0  70EF                MDX     NEXT1
  001B 30 14162897      ERR2  CALL    MESSP
  001D 1  005E                DC      ERR2P
  001E 30 25241600            CALL    VIAQ
  0020 0  0002          AREA1 DC      2
  0021 0  1001                DC      /1001
  0022    0001                BSS     1
  0023 0  0000                DC      /0000
  0024 0  0004          AREA2 DC      4
  0025 0  1001                DC      /1001
  0026    0002                BSS     2
  0028 0  AAAA                DC      /AAAA
  0029    0020          ERR1P EBC     .    NOT SUCCESSFUL, EITHER CARD.
  0039    0019                EBC     . WITH BAD FORMAT WAS NOT .
  0046    001C                EBC     .RECOGNIZED OR TOO MANY DATA .
  0054    0012                EBC     .CARDS WERE READ IN.
  005D    0002                EBC     .$$.
  005E    001D          ERR2P EBC     .    NOT SUCCESSFUL, I/O AREA.
  006D    001D                EBC     . ZEROED ABOVE AREA FOR WHICH .
  007C    0011                EBC     .DATA WAS SUPPLIED.
  0085    0002                EBC     .$$.
  0086 0  AAAA          CONST DC      /AAAA
  0088    0000                END     START

     NO ERRORS IN ABOVE ASSEMBLY.
DB638
DUP FUNCTION COMPLETED
// DUP
*SIMULCIL M         DB638 DB638 DB638
*CCEND

 CLB, BUILD DB638

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0054
 *IST TABLE 3F0F  0036
 *PNT TABLE 3F46  0008
 MAIN DB638 3F4E
 PNT  DB638 3F48
 PNT  DB638 3F4C
 CALL VIAQ  3FD6
 CALL MESSP 4036
 CORE       40BE  3F42
 CLB, DB638 LD XQ

*XIO
*START SIMULATION
S 50    3F4E  AISQN
S 20 CONTROL WORD!1000, IO AREA!3F6E.SPECIAL ENTRY !3000
S 14 INPUT CARD ! *AI   ƀ11120-11134ƀ02561-19825ƀ31562
S 12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
S 14 INPUT CARD ! *AI   12340210FF12FEDC
S 12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
S 14 INPUT CARD ! *AI   00001010010001100000011111111111
S 12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
S 14 INPUT CARD ! *AI D ƀ00123-00010ƀ00127-02047
S 15 TOO MUCH DATA, READ CARDS UNTIL E IN COL 72
S 14 INPUT CARD ! *AI H 12340210FF12FEDC                                         E
S 50    3F52  AISQN
S 21 BUSY TEST
S 50    3F52  AISQN
S 21 BUSY TEST
S 50    3F58  AISQN
S 20 CONTROL WORD!1000, IO AREA!3F72.SPECIAL ENTRY !3000
S 14 INPUT CARD ! *AI H 1234FF12    FEDC                                         E
S 16 INSUFFICIENT DATA TO FILL I/O AREA
S 50    3F5C  AISQN
S 21 BUSY TEST
S 50    3F5C  AISQN
S 21 BUSY TEST
S 56 TSX  VIAQ   NOT EXECUTED
S 99  0015  3F63  3F65  5400  3FD6  0000  0000  0000  0000  0000  3F38

N08 ILLEG LDR CD
*END DATA
// *                DB638 END
```

EXAMPLE 9 (PROGRAM LISTING NO. 3). The Simulator control cards are read in and initializing processes are begun. The three analog input cards signify card input for data. Note that one or all of these cards could have specified the random number generator as an input source. Note also that in a program such as this, extreme care must be taken in setting up the data cards, remembering that an E parameter in column 72 terminates any attempt to fill an analog buffer.

The S56 message reflects the call in the FORTRAN program. Note that the parameters are also printed. If either of these parameters had been improper, a message would be printed accordingly.

The next block of four PRNTN calls refer to the WRITE (M, 10) statement. A FORTRAN call to output or input, generally, translates to several calls to the I/O subroutine, including busy tests, actual I/O, and special functions as in the carriage control shown here. The line after S33 gives the message which the user would see if the program were executed.

The three CARDN calls listed next are the result of the FORTRAN statement 20 READ (N, 30). The card image in hexadecimal is printed and the busy tests performed.

Next, in sequence, are four print calls: the result of the FORTRAN statement 35 WRITE (M, 40). Again, the line after S33 gives the users actual printout.

Note that the FORTRAN statement at 30+1 is an IF statement. The Simulation output gives no indication of this because no major subroutines are called; that is, the Simulator does not show when some arithmetic function or subroutine is called. However, all instructions of this statement have been simulated.

The first DAOP call in the Simulator output is the result of the FORTRAN CALL DAC statement. Again, a rundown of the calling sequence is given in S20. The S10 message describes the type of output, that is, random, sequential, etc. The four words of analog output are given just ahead of the next S50 message. The DAOP busy tests are a result of the second DAC call.

Next, in sequence, is a series of analog input calls, each one reading some point from cards. These are a direct result of the analog input calls in the FORTRAN program.

Again, there are four calls to the PRNTN routine: the result of WRITE (M, 65).

At this point, the IF following statement 65 and the IF at statement 90 force a return to statement 20 and a second "go" through the loop is simulated. The message "GOT THIS FAR" is printed out a second time, and for a third time, statement 20 is executed and a card is read. At this point, the IF at 30+1 forces a transfer to statement 120.

The CALL QUEUE, CALL SHARE, and CALL VIAQ are then simulated by the S56 messages. Note that in the case of CALL QUEUE, the name of the called program TC152 is also printed out. A snapshot of the terminating instruction is then printed.

Since the user included a *DUMP control card, the program is dumped. The addresses are absolute. The address in statement S98 gives the absolute address of the first word of the user's program. The XXXX at the end of the dump refers to undefined core.

A dump of the transfer vector may be obtained by using a negative number as the lower limit.

One method of tracing through a FORTRAN program is by strategic PAUSE (I) statements. When the Simulator encounters such a statement, it will print out the S56 PAUSE message together with the appropriate parameter. Thus, some idea of program flow may be obtained.

PROGRAM LISTING NO. 3 -- EXAMPLE 9

```
// JOB         A
// *                                                 SIMULATOR TEST CASE 152
// FOR TC152
*LIST ALL                                                                       00000000
*IOCS(1443 PRINTER  )                                                           00000010
*ONE WORD INTEGERS                                                              00000030
**   SIMULATOR TEST CASE 152
      EXTERNAL TC152
      DIMENSION NOUTA(10),IN1(9),IN2(10),ID1(10)                                00000050
      CALL UNMK(-1,-1)
      IXIT=0
      N = 2                                                                     00000060
      M = 3                                                                     00000070
      ITOL = 20                                                                 00000080
      WRITE (M,10)                                                              00000090
   10 FORMAT(24H1INTERLEAVED AIP,AIS,AIR)                                       00000100
   20 READ(N,30) (NOUTA(I),I=1,4),IN1(1),IN2(1),IXIT
   30 FORMAT(6I6,I1)
      IF(IXIT) 120,35,120
   35 WRITE (M,40) (NOUTA(I),I=1,4),IN1(1),IN2(1)                               00000130
   40 FORMAT (1H,6I7)                                                           00000140
      CALL DAC (01101,NOUTA(1),NOUTA(5))                                        00000150
   50 CALL DAC (0,J)                                                            00000160
      GOTO (50,60),J                                                            00000170
   60 CALL AIP (01000,JP,IN1(1))
      CALL AIS (02001,ID1(1),ID1(3),IN2(1))
      CALL AIR (02001,ID1(1),ID1(2),IN2(1),IN2(1),0□                            00000200
      CALL AIS (02001,ID1(1),ID1(3),IN2(1))
      CALL AIR (02001,ID1(1),ID1(2),IN2(1),IN2(1),0□
      CALL AIP (01000,JP,IN1(1))
      WRITE(M,65)
   65 FORMAT(13H GOT THIS FAR)                                                  00000250
      IF (IABS(ID1(1)-NOUTA(3))-ITOL) 90,90,70                                  00000260
   70 WRITE (M,80) ID1(1)                                                       00000270
   80 FORMAT (10H ID1(1) = ,I7/17H OUT OF TOLERANCE□                            00000280
   90 IF (IABS(NOUTA(1)-JP)-ITOL)  20,20 ,100                                   00000290
  100 WRITE (M,110) JP                                                          00000300
  110 FORMAT ( 6H JP = ,I7/17H OUT OF TOLERANCE)                                00000310
      GO TO 20                                                                  00000320
  120 CALL QUEUE(TC152,1,5)
      CALL SHARE(300)
      CALL VIAQ
  130 GO TO 130
      END                                                                       00000330

VARIABLE ALLOCATIONS
 NOUTA=000D  IN1  =0016  IN2  =0020  ID1  =002A  IXIT #002B  N    #002C  M    #002D  ITOL #002E  I    =002F  J    =0030
 JP   =0031

STATEMENT ALLOCATIONS
 10   =004D  30   =005B  40   =005F  65   =0064  80   #006D  110  #0080  20   #0083  35   #00D6  50   =0106  60   =0110
 70   =01BB  90   =01C3  100  =01D2  120  =01DA  130  #01E5

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 TC152   UNMK    DAC     AIP     AIS     AIR     IABS    QUEUE   SHARE   VIAQ    COMGO   MRED    MWRT    MCOMP   MIOIX
 MIOI    SUBSC   PRNTN   EBPRT

INTEGER CONSTANTS
     1=0042      0=0043      2=0044      3=0045      20#0046      4#0047   1101#0048   1000#0049   2001=004A      5=004B
   300=004C

CORE REQUIREMENTS FOR TC152
 COMMON      0  INSKEL COMMON      0  VARIABLES     66  PROGRAM    422


 END OF COMPILATION


TC152
DUP FUNCTION COMPLETED
// DUP
*DELET     M          TC152
TC152
D25 NAME NOT IN L/F
*SIMULCIL M           TC152 TC152 TC152
*INCLDTRACE/2800
*CCEND

CLB, BUILD TC152

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2
```

```
 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0054
 *VTV TABLE 3F0F  0006
 *IST TABLE 3F15  0036
 *PNT TABLE 3F4C  000C
 MAIN TC152 3FE9
 PNT  TC152 3F4E
 PNT  TC152 3F52
 CALL UNMK  4140
 CALL AIP   418A
 CALL AIR   41B8
 CALL QUEUE 42CA
 PNT  TC152 3F56
 CALL VIAQ  438C
 LIBF AIPTN 43EC  3F0F
 LIBF AIRN  446E  3F12
 CORE       4544  3ABC

 CLB, TC152 LD XQ

*AIP C
*AIR C
*AIS C
*XIO
*DUMP 0000 7FFF
*START SIMULATION
S 56 TSX  UNMK   NOT EXECUTED
       DC     FFFF
       DC     FFFF
S 50     293A  PRNTN
S 34-0 LIST PRNTR CARRIAGE CONTROL WORD IS /3100
S 50     293C  PRNTN
S 33-0 LIST PRNTR OUTPT CONTROL WORD : /2110
INTERLEAVED AIP,AIS,AIR
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2928  CARDN
S 27-0 CARD ,INPUT, WORD COUNT :     80
0000  0000  0000  0000  1000  2000  0000  0000  0000  0000  0000  2000  0000  0000  0000  0000  0800  2000  0000  0000
0000  0000  0000  1000  0000  0000  0200  2000  0010  0080  0000  0000  0200  2000  0010  0040  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
S 50     292C  CARDN
S 21-0 BUSY TEST
S 50     292C  CARDN
S 21-0 BUSY TEST
S 50     293A  PRNTN
S 34-0 LIST PRNTR CARRIAGE CONTROL WORD IS /3D00
S 50     293C  PRNTN
S 33-0 LIST PRNTR OUTPT CONTROL WORD : /2110
     10       0      20       1    4096    4097
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     1A48  DAOP
S 20 CONTROL WORD:1010 ,I/O AREA ADDRESS: 3F61 ,SPECIAL RETURN ADDRESS! 3F11
S 10  WRITE OUTPUT  - - - - - RANDOM ADDRESSING
           WORD COUNT:/0004
0001  0014  0000  000A
S 50     1A48  DAOP
S 21 BUSY TEST
S 50     1A48  DAOP
S 21 BUSY TEST
S 50     1A48  AIPTN
S 20 CONTROL WORD:1000 ,AREA :3F89 ,MULTIPLEXER ADDRESS : 0201
S 14 INPUT CARD : *AI D &00010                                                          E
S 50     1A48  AISQN
S 20 CONTROL WORD:2000, IO AREA:3F80,SPECIAL ENTRY :0000
S 14 INPUT CARD : *AI D &00020                                                          E
S 50     1A48  AIRNN
S 20 CONTROL WORD:1000 ,IO AREA ADDRESS:3F81,MULTIPLEXER TABLE ADDRESS! 3F78,RELAY ADDR:0000
S 14 INPUT CARD : *AI D &00020                                                          E
S 50     1A48  AISQN
S 20 CONTROL WORD:2000, IO AREA:3F80,SPECIAL ENTRY :0000
S 14 INPUT CARD : *AI D &00020                                                          E
S 50     1A48  AIRNN
S 20 CONTROL WORD:1000 ,IO AREA ADDRESS:3F81,MULTIPLEXER TABLE ADDRESS! 3F78,RELAY ADDR:0000
S 14 INPUT CARD : *AI D &00020                                                          E
S 50     1A48  AIPTN
S 20 CONTROL WORD:1000 ,AREA :3F89 ,MULTIPLEXER ADDRESS : 0201
S 14 INPUT CARD : *AI D &00010                                                          E
```

```
S 50     293A  PRNTN
S 34-0 LIST PRNTR CARRIAGE CONTROL WORD IS /3D00
S 50     293C  PRNTN
S 33-0 LIST PRNTR OUTPT CONTROL WORD : /2110
GOT THIS FAR
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2928  CARDN
S 27-0 CARD  INPUT, WORD COUNT :     80
0000  0000  0000  0000  1000  2000  0000  0000  0000  0000  0000  2000  0000  0000  0000  0000  0800  2000  0000  0000
0000  0000  0000  1000  0000  0000  0200  2000  0010  0080  0000  0000  0200  2000  0010  0040  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
S 50     292C  CARDN
S 21-0 BUSY TEST
S 50     292C  CARDN
S 21-0 BUSY TEST
S 50     293A  PRNTN
S 34-0 LIST PRNTR CARRIAGE CONTROL WORD IS /3D00
S 50     293C  PRNTN
S 33-0 LIST PRNTR OUTPT CONTROL WORD : /2110
     10       0      20       1    4096    4097
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     1A48  DAOP
S 20 CONTROL WORD:1010 ,I/O AREA ADDRESS: 3F61 ,SPECIAL RETURN ADDRESS: 3F11
S 10  WRITE OUTPUT  - - - - - RANDOM ADDRESSING
            WORD COUNT:/0004
0001  0014  0000  000A
S 50     1A48  DAOP
S 21 BUSY TEST
S 50     1A48  DAOP
S 21 BUSY TEST
S 50     1A48  AIPTN
S 20 CONTROL WORD:1000 ,AREA :3F89 ,MULTIPLEXER ADDRESS : 0201
S 14 INPUT CARD : *AI D &00010                                                 E
S 50     1A48  AISQN
S 20 CONTROL WORD:2000, IO AREA:3F80,SPECIAL ENTRY :0000
S 14 INPUT CARD : *AI D &00020                                                 E
S 50     1A48  AIRNN
S 20 CONTROL WORD:1000 ,IO AREA ADDRESS:3F81,MULTIPLEXER TABLE ADDRESS: 3F78,RELAY ADDR:0000
S 14 INPUT CARD : *AI D &00020                                                 E
S 50     1A48  AISQN
S 20 CONTROL WORD:2000, IO AREA:3F80,SPECIAL ENTRY :0000
S 14 INPUT CARD : *AI D &00020                                                 E
S 50     1A48  AIRNN
S 20 CONTROL WORD:1000 ,IO AREA ADDRESS:3F81,MULTIPLEXER TABLE ADDRESS: 3F78,RELAY ADDR:0000
S 14 INPUT CARD : *AI D &00020                                                 E
S 50     1A48  AIPTN
S 20 CONTROL WORD:1000 ,AREA :3F89 ,MULTIPLEXER ADDRESS : 0201
S 14 INPUT CARD : *AI D &00010                                                 E
S 50     293A  PRNTN
S 34-0 LIST PRNTR CARRIAGE CONTROL WORD IS /3D00
S 50     293C  PRNTN
S 33-0 LIST PRNTR OUTPT CONTROL WORD : /2110
GOT THIS FAR
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2940  PRNTN
S 21-0  LIST PRINTER BUSY TEST
S 50     2928  CARDN
S 27-0 CARD  INPUT, WORD COUNT :     80
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  1000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
S 50     292C  CARDN
S 21-0 BUSY TEST
S 50     292C  CARDN
S 21-0 BUSY TEST
S 56 TSX  QUEUE  NOT EXECUTED
         CALL   TC152
         DC     0001
         DC     0005
S 56 TSX  SHARE  NOT EXECUTED
         DC     012C
S 56 TSX  VIAQ   NOT EXECUTED
S 99  01E3  413B  413D  5400  438C  0001  0000  0000  0000  28D3  3F3B
S 98  DUMP OF SIMULATED CORE  3F58
```

```
3F50                                                    FFF2  0000  FFFF  0000  0000  0000  0000  0000
3F60    0000  0004  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
        3F70  TO  3F7F  CONTAINS  0000
3F80    0002  0001  0028  0001  0002  0003  0014  0005  0002  0018  FFFD  0000  0000  0000  0000  0000
3F90    0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0001  0000  0002  0003  0014  0004
3FA0    044D  03E8  07D1  0005  012C  5018  F1C9  D5E3  C5D9  D3C5  C1E5  C5C4  40C1  C9D7  6BC1  C9E2
3FB0    6BC1  C9D9  B00D  2006  9006  2001  B003  5001  6B40  2007  9006  B004  500D  40C7  D6E3  40E3
3FC0    C8C9  E240  C6C1  D940  B008  500A  40C9  C4F1  4DF1  5D40  7E40  2007  7000  5011  40D6  E4E3
3FD0    40D6  C640  E3D6  D3C5  D9C1  D5C3  C540  B012  5006  40D1  D740  7E40  2007  7000  5011  40D6
3FE0    E4E3  40D6  C640  E3D6  D3C5  D9C1  D5C3  C540  B010  1010  9400  3F9A  D400  3F58  1010  9400
3FF0    3F9A  D400  3F5A  5400  4140  3F58  3F5A  C400  3F9B  D400  3F83  C400  3F9C  D400  3F84  C400
4000    3F9D  D400  3F85  C400  3F9E  D400  3F86  5392  3F85  3FA5  5395  53AA  3F84  3FB3  C400  3F9A
4010    D400  3F87  53A1  3F8A  FFFF  3F87  8001  539E  3F65  7401  3F87  C400  3F87  9400  3F9F  4C08
4020    4012  6500  0000  539E  3F6E  539E  3F78  539B  3F83  5395  C400  3F83  4C20  4132  5392  3F85
4030    3FB7  C400  3F9A  D400  3F87  53A1  3F8A  FFFF  3F87  8001  539E  3F65  7401  3F87  C400  3F87
4040    9400  3F9F  4C08  4035  6500  0000  539E  3F6E  539E  3F78  5395  6500  0000  7500  3F65  7000
4050    6D00  405C  6500  FFFC  7500  3F65  7000  6D00  405D  5480  3E36  3FA0  3F65  3F61  5480  3E36
4060    3F9B  3F88  6580  3F88  53A4  0002  405E  4068  6500  0000  7500  3F6E  7000  6D00  4073  5400
4070    418A  3FA1  3F89  3F6E  6500  0000  7500  3F82  7000  6D00  408C  6500  FFFE  7500  3F82  7000
4080    6D00  408D  6500  0000  7500  3F78  7000  6D00  408E  5480  3E35  3FA2  3F82  3F80  3F78  6500
4090    0000  7500  3F82  7000  6D00  40AE  6500  FFFF  7500  3F82  7000  6D00  40AF  6500  0000  7500
40A0    3F78  7000  6D00  40B0  6500  0000  7500  3F78  7000  6D00  40B1  5400  41B8  3FA2  3F82  3F81
40B0    3F78  3F78  3F9B  6500  0000  7500  3F82  7000  6D00  40CB  6500  FFFE  7500  3F82  7000  6D00
40C0    40CC  6500  0000  7500  3F78  7000  6D00  40CD  5480  3E35  3FA2  3F82  3F80  3F78  6500  0000
40D0    7500  3F82  7000  6D00  40ED  6500  FFFF  7500  3F82  7000  6D00  40EE  6500  0000  7500  3F78
40E0    7000  6D00  40EF  6500  0000  7500  3F78  7000  6D00  40F0  5400  41B8  3FA2  3F82  3F81  3F78
40F0    3F78  3F9B  6500  0000  7500  3F6E  7000  6D00  40FD  5400  418A  3FA1  3F89  3F6E  5392  3F85
4100    3FBC  5395  6500  0000  C500  3F82  6500  FFFE  9500  3F65  D400  3F58  5480  3E34  3F58  9400
4110    3F86  4C08  411B  5392  3F85  3FC5  6500  0000  539E  3F82  5395  6500  0000  C500  3F65  9400
4120    3F89  D400  3F58  5480  3E34  3F58  9400  3F86  4C08  400B  5392  3F85  3FD8  539B  3F89  5395
4130    4C00  400B  5400  42CA  5400  3F56  3F9A  3FA3  5480  3E3E  3FA4  5400  438C  4C00  413D  0000
4140    0000  0C00  0032  0C00  0034  6934  6A35  6100  6680  4140  C132  F680  0000  D039  C134  F680
4150    0001  D036  C164  4C18  416B  C680  0000  E16C  D173  C680  0001  E16D  E973  4C18  416B  C027
4160    E96C  D025  C025  E96D  D023  C0DA  D174  C021  D173  4480  0078  C12E  E01A  D12E  C16E  E017
4170    D16E  C130  E015  D130  C16F  E012  D16F  7402  4140  6500  0000  6600  0000  0C00  002E  0C00
4180    0030  0C00  00A0  0C00  00A2  4C80  4140  0000  0000  0083  40FB  4480  009A  C824  5480  3E2B
4190    4820  7019  C33D  E01D  901F  4818  700E  6680  0036  C200  D33E  C201  D33F  7402  0036  C80E
41A0    DB40  C00E  D342  4F00  003A  C009  D33E  C008  D33F  4F00  003A  C007  5480  3E2A  0000  4480
41B0    009B  F000  53D4  003C  3000  0000  53D7  003C  40EC  4480  009A  C8FA  5480  3E2B  D400  0037
41C0    D342  9052  D343  C33D  E058  9054  4C18  426A  C342  4808  7032  7401  0036  6680  0036  C200
41D0    D340  8042  D341  C400  0037  1001  8400  0036  D400  0036  6580  0036  C33D  E039  D400  0036
41E0    C33D  E03A  D33D  C340  7400  0036  802D  D33E  C100  D344  7400  0036  8027  D33F  91FF  4830
41F0    700A  C2FF  9200  4808  7006  E828  D345  7400  0036  7006  7032  6E00  0036  C0B9  5480  3E2A
4200    C200  D680  0000  C100  D580  0000  C345  900D  D780  0041  74FF  0037  7015  C340  D680  FFFF
4210    C344  D580  FFFF  702F  0001  0002  0FFF  0020  1000  3000  4000  5000  FF1F  F000  C000  0000
4220    4480  009B  7202  C200  D680  FFFD  80ED  D341  7102  C100  D580  FFFD  70C1  C343  4818  7042
4230    C342  4820  7044  C400  0037  90DE  4820  70C8  C200  D680  0000  C100  D580  0000  C345  90D4
4240    E0D5  D780  0041  C33D  E0D5  4818  7007  C101  D342  C0D6  D343  C0D5  D344  7002  C8D1  DB42
4250    C33D  E0C6  4820  702C  1010  D341  7102  6D00  0036  1010  D340  C33D  E0C0  90BC  4810  7006
4260    C33D  E0B4  E8B5  D33D  4F00  003A  C33D  E0AE  E8B0  70F9  C8B5  DB3E  C33D  E0A8  E8AC  D33D
4270    4F00  003A  C345  E0A2  D680  0000  70CC  1804  D342  C345  9099  D680  0000  74FF  0037  70A2
4280    C33D  4804  7011  6780  0067  53D7  0000  70FD  C03B  6680  42C4  6780  0068  D341  C102  D20A
4290    C103  D20B  7102  70C2  6780  0067  53D7  0001  70FD  C02B  6680  42C5  70EE  0000  1010  1082
42A0    D00C  1081  4804  700B  1010  108D  D007  4400  0000  42AD  42AE  4C80  429D  0000  0000  C019
42B0    D0FC  70F2  0000  1010  1082  D00C  1081  4804  700D  1010  108D  D007  4400  0000  42C2  42C3
42C0    4C80  42B2  0000  0000  429D  42B2  C002  D0FA  70F0  0003  0000  4480  009A  7401  0036  C400
42D0    0036  807A  D33A  C780  003A  D342  C480  0036  D33D  CF80  003D  4C08  42FE  DB3E  6680  007A
42E0    C400  007A  A068  1090  8400  0079  D400  0037  D341  1810  D343  6580  0037  C100  4C18  4302
42F0    B33F  7015  7014  407C  C100  B33F  7057  7056  C780  0042  71FE  B100  706D  706C  7403  0036
4300    4480  009B  6D00  0037  C400  0037  D343  71FD  72FF  70E3  4065  C343  4C18  4377  C780  0043
4310    4C20  434E  C343  9039  D343  C341  9036  D341  C343  D400  0037  6680  0037  0C00  002E  0C00
4320    0030  6E00  0037  C341  9400  0037  4C08  4353  7203  C780  0042  B200  70F4  70F3  4041  B200
4330    701D  701C  C780  0043  4C20  434E  C343  D400  0037  6103  C200  D480  0037  1810  D200  7201
4340    7401  0037  71FF  70F6  72FD  6E00  0037  C400  0037  D343  70D2  0003  0001  0002  0C00  002E
4350    0C00  0030  708B  401C  C780  0043  4C20  434E  C343  D400  0037  C780  0042  D480  0037  7401
4360    0037  C33E  D480  0037  7401  0037  C33F  D480  0037  7094  0C00  002E  0C00  0030  7102  7097
4370    0000  0C00  0032  0C00  0034  4C80  4370  6100  C780  0042  D176  CB3E  D974  C00D  D173  C33A
4380    80CB  D343  C780  0043  D343  C780  0043  D177  4480  0078  70DE  0078  0000  6580  0079  7101
4390    6680  007A  C100  D054  4C18  43C9  4047  6951  C100  D04E  7102  C100  D04D  71FE  4038  72FF
43A0    7001  700B  7103  C100  4C18  439F  C041  4036  B100  70EC  1000  402B  70F2  4030  6680  43E9
43B0    C200  9036  4C20  438D  7202  C200  9033  4C20  438D  C200  D02C  1810  D200  72FF  C200  D026
43C0    1810  D200  72FF  D200  4012  4480  006A  0000  43E6  7103  72FF  70C6  C400  0029  4C08  43D5
43D0    D0F6  5480  3E3E  43C7  70B8  3000  70B6  0000  0C00  002E  0C00  0030  4C80  43D7  0000  0C00
43E0    0032  0C00  0034  4C80  43DE  0000  0000  0000  0000  0000  0000  0000  0000  4480  00AC  5480
43F0    3E29  7400  4402  700A  C02C  1008  4C10  43FB  C480  4422  7001  C026  D005  D058  C480  0037
4400    617F  6600  0000  4804  721E  1804  D346  1804  D347  1804  4C20  4415  C204  4C20  4411  7401
4410    0037  7401  0037  5480  3E28  1801  4C04  445B  617F  09B3  09B5  C204  4C08  4424  09AF  09B1
4420    70F7  5480  3E44  FFFD  C201  E113  D201  1010  D219  C346  100F  1808  8201  D201  C347  100E
4430    180B  8201  D201  7401  0037  C480  0037  D20B  7401  0037  C480  0037  D200  C680  0000  1003
4440    4C28  4444  C105  7001  1010  8106  D20A  E106  D204  C1B8  D21C  C1E9  D217  7401  0037  C480
4450    0037  D213  0A00  C20B  4480  0062  0000  D21A  C0CA  D214  70B6  0A14  C204  4C18  4464  C21A
4460    4C08  4464  74FF  0007  7000  1010  D20A  D20C  D211  D204  D20F  D219  D21A  70A3  0000  4480
4470    00AC  5480  3E29  7400  4485  700B  C021  1008  4C10  447D  C480  4499  7001  C01B  D006  D400
4480    4529  617F  C480  0037  6600  0000  4804  721E  1804  D34C  1804  D34D  1804  4C20  449A  C204
4490    4C20  4494  7401  0037  7401  0037  5480  3E28  5480  3E44  D1B7  1010  D34E  D34F  C03E  6580
44A0    0036  71FB  703C  4D80  44A9  44AB  44AA  44AE  44AD  452D  D34F  D34E  7001  D34F  617F  09B3
44B0    09B5  C204  4C08  44B7  09B1  09B1  70F7  C34F  D219  7401  0037  C480  0037  D001  6500  0000
```

```
44C0    6918  C100  1001  4C02  44C8  4C28  44E5  7020  1001  1802  D001  7500  0000  7101  C100  F580
44D0    0000  4C20  44DF  C100  8008  D004  9005  4C18  44E8  6500  0000  70E5  0000  0001  0020  613C
44E0    6680  0037  4C80  0072  8000  1001  1802  D100  617F  C480  0037  D206  C207  E113  E8EF  D207
44F0    C34C  100F  1808  8207  D207  C34D  100E  1808  8207  D207  7401  0037  C480  0037  D208  7401
4500    0037  C480  0037  D213  7401  0037  C480  0037  D218  C1B8  D21C  C1E9  D217  C34E  4C04  451F
4510    C680  0018  4C20  4519  613C  6680  0037  4C80  0072  7401  0037  C480  0037  D212  D20B  0A06
4520    0A08  C00A  D214  C089  D204  D211  C206  4480  0062  0000  D21A  7013  FFFD  0A14  C204  4C18
4530    4536  C21A  4C08  4536  74FF  0007  7000  1010  D20A  D20C  D211  D204  D20F  D219  D21A  4C00
4540    4494  0000  0000  0000  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX
        4550  TO  BF4F  CONTAINS  XXXX
BF50    XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX  XXXX

N04 READY READER
```

EXAMPLE 10. RESERVING A FILE AREA IN THE CORE LOAD AREA. The only method of reserving a file area in the Core Load Area on disk is through an *STOREDATA function. Depending on the intent and purpose of the user at the time this function is performed, two options are open to him:

1. He may wish to reserve (that is, set aside) a file area in the Core Load Area for future use. The source input may be data from cards (D in column 11 and RD in columns 13-14) which is stored directly to its destination area (FX in columns 17-18), or it may be information in the Nonprocess Working Storage (D in column 11 and FX in columns 17-18) which is physically moved to a data block that starts at the beginning of the next available sector in the Core Load Area. Note that in both cases, the input may be valid or invalid information.

   For example, if the data cards contain zeros throughout, a new file area will be established.
2. He may elect to move or transport a data block of true or valid information from the start of Nonprocess Work Storage into the Core Load Area.

To reserve a data file, therefore, implies that there is movement of information or data which may be valid or invalid. Note that the *STOREDATA function transfers this information to disk without any change of format.

To accomplish this, certain parameters must be known to DUP and the Core Load Builder. These are:

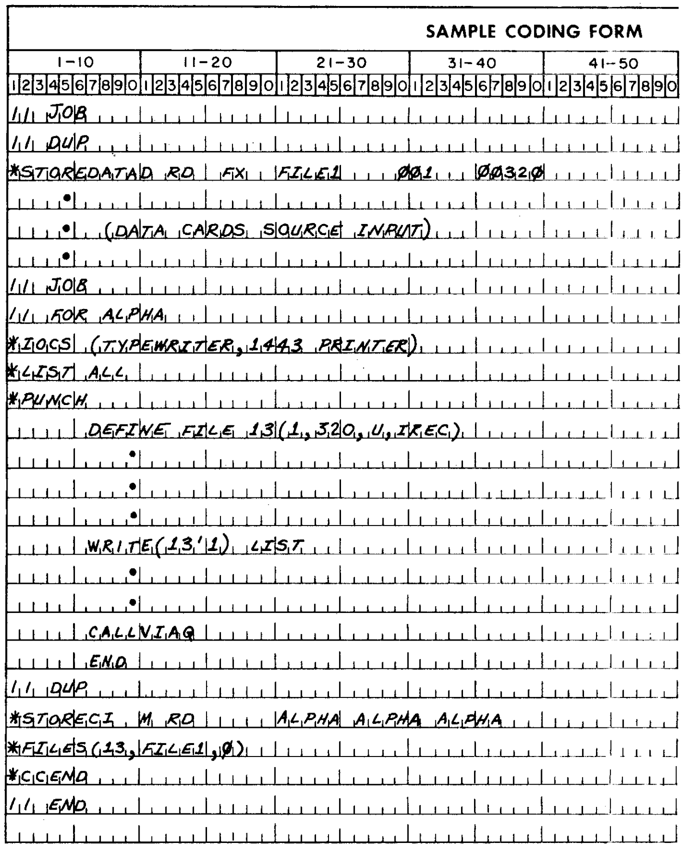
- Type of source input
- Logical drive number
- Destination area
- Name of data file
- Sector count and/or word count

Figure 39 gives the sequence of control cards that might be used in a typical situation, where the source input is data cards.

A data file, FILE1, one sector long, is reserved in the Core Load Area, and given a FLET entry. Since this is a data FLET entry, the control card sector count, or the sector equivalent of the word count, is contained in the entry.

The DEFINE FILE statement (see IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715) specifies to the FORTRAN Compiler the size and quantity of disk data records within a file (or files) that will be used for processing with a particular program and its associated programs; in this example, one data block of 320 words is used. Since we are storing data to be used by a FORTRAN program from cards, the associated *FILES control record must contain the identical name (that is, FILE1) of the data block established by the *STOREDATA function. Note that it is from the *FILES card that the Core Load Builder obtains the necessary data (name, file number and drive code) to construct the DEFINE FILE TABLE within a core load. This is a three-word table which equates program defined file numbers to symbolic disk data files specified on the *FILES card.

We have seen that in moving data from the Nonprocess Working Storage to the Core Load Area, the direction of movement is from the start of Nonprocess Working Storage on the disk drive specified.



SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
1234567890123456789012345678901234567890123456789 0
// JOB
// DUP
*STOREDATAD RD  FX  FILE1    ØØ1   ØØ320
    .
    .  (DATA CARDS SOURCE INPUT)
    .
// JOB
// FOR ALPHA
*IOCS (TYPEWRITER,1443 PRINTER)
*LIST ALL
*PUNCH
      DEFINE FILE 13(1,320,U,IREC)
         .
         .
         .
      WRITE(13'1) LIST
         .
         .
      CALLVIAQ
      END
// DUP
*STORECI  M RD      ALPHA ALPHA ALPHA
*FILES(13,FILE1,Ø)
*CCEND
// END
```

Figure 39. Reserving a File Area in the Core Load Area

However, in the case of data files referenced by FORTRAN I/O in Nonprocess Working Storage, the direction of movement starts from the end of Nonprocess Working Storage, and therefore the movement of data through the *STOREDATA function does not actually move a data file established by the execution of a FORTRAN-written program.

EXAMPLE 11. DUMPING A PROGRAM OR DATA FILE FROM THE CORE LOAD AREA. A program or a block of data in the Core Load Area may be

1. moved to the Nonprocess Working Storage,
2. punched into cards, or
3. printed on the List Printer.

Figure 40 illustrates these three cases.

The primary difference between the two dump functions, *DUMP and *DUMPDATA, lies in the handling of relocatable programs. *DUMP converts relocatable programs from disk system format either to card system format when dumping to cards, or to printer format when the List Printer is selected as the I/O media. *DUMPDATA performs no conversion and outputs a relocatable program as data identical to its original format.

Programs and/or data in the Core Load Area are assumed to be in disk core image format. The name of the program or data file must always be given.

Note that when dumping from the Core Load Area to punched cards, an *CCEND control card is punched out as the last card in the output deck. This card also contains the word and sector count needed in a *STOREDATA operation. (Note also that, although the *STOREDATA function requires that these counts be contained in the control card, the punched-out *CCEND card may be used in a subsequent store of the dumped core load.) PN specifies the Card Punch as the principal system output device, while PR specifies the List Printer.

In the same fashion, it is also possible to dump a mainline, combination, or interrupt core load to Nonprocess Working Storage or to any available I/O media.

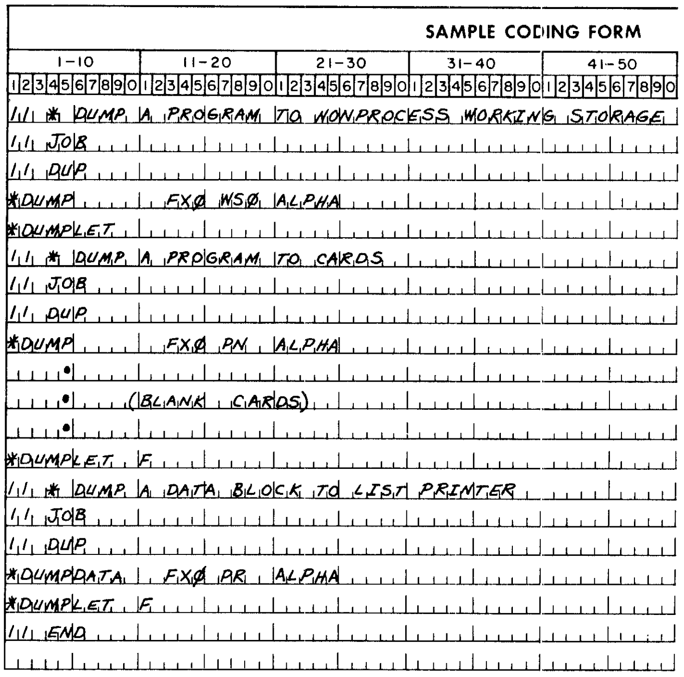
EXAMPLE 12. LOADING A PROGRAM OR DATA BACK INTO THE CORE LOAD AREA. One of the features of the Disk Utility Program (DUP) is the ability to load (that is, store) a previously built core load, which has been dumped to cards by the *DUMPDATA function, back to the Core Load Area. One significant use of this ability is the reordering (that is, rearrangement) of the position of core loads within the Core Load Area.

Consider the following example. Core loads ALPHA, BETA1, DELTA, and GAMMA reside in this order-sequence in the Core Load Area. It is desired, for chaining purposes, to use them at online time in some other arrangement: say, GAMMA, BETA1, ALPHA, and DELTA. The four core loads are first dumped to cards by a series of *DUMPDATA operations, and then deleted from the Core Load Area. A reload of the new sequence of core loads is now performed in the order desired. The result is a greater efficiency in the usage of the disk by the reduction of disk seek time.

Figure 41 illustrates a possible card deck arrangement for this situation.

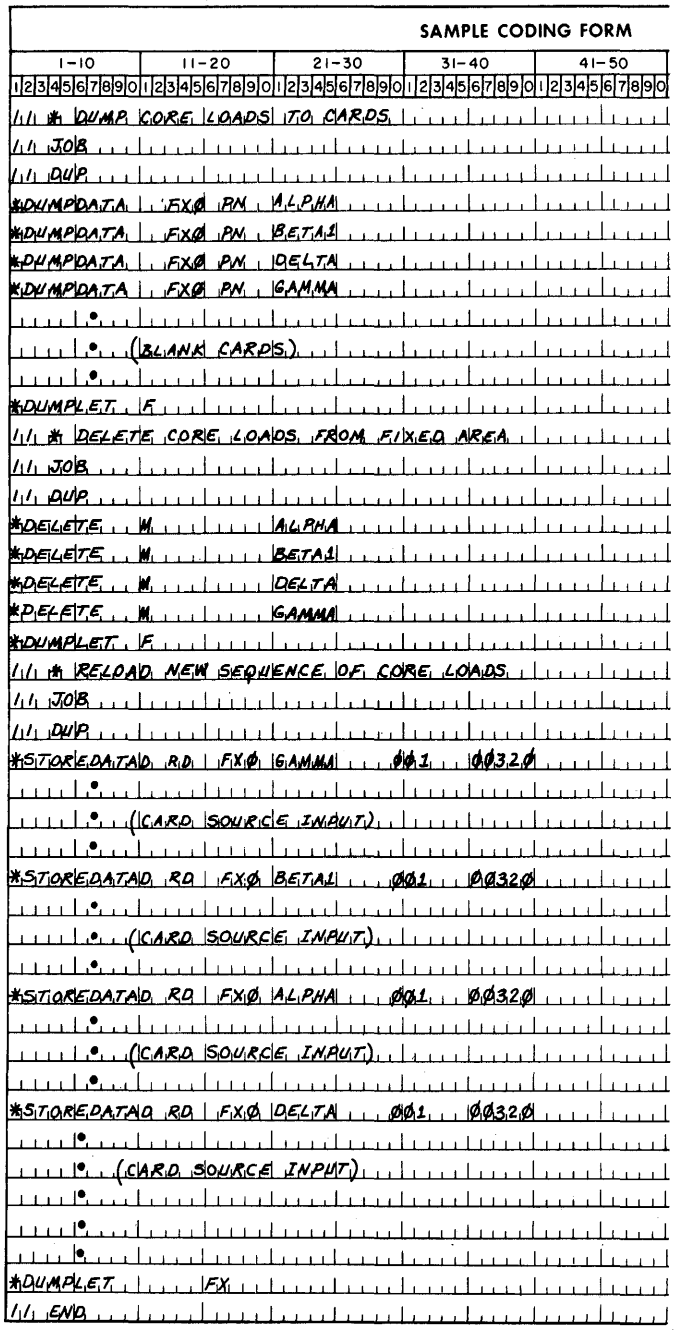
EXAMPLE 13. DUMPING A PROGRAM FROM THE RELOCATABLE PROGRAM (OR USER) AREA. A dump of a user-written or IBM program may be made from the User Area to any of the following I/O media:

1. Nonprocess Working Storage (NPWS)
2. Punched cards
3. List Printer


SAMPLE CODING FORM

```
         1-10       11-20      21-30      31-40      41-50
      1234567890 1234567890 1234567890 1234567890 1234567890
// * DUMP A PROGRAM TO NONPROCESS WORKING STORAGE
// JOB
// DUP
*DUMP           FXØ WSØ ALPHA
*DUMPLET
// * DUMP A PROGRAM TO CARDS
// JOB
// DUP
*DUMP           FXØ PN  ALPHA
    •
    •    (BLANK CARDS)
    •
*DUMPLET  F
// * DUMP A DATA BLOCK TO LIST PRINTER
// JOB
// DUP
*DUMPDATA       FXØ PR  ALPHA
*DUMPLET  F
// END
```

Figure 40. Illustrating Various Card Arrangements in Dumping a Program/Data to Nonprocess Working Storage, Punched Cards, and the List Printer



SAMPLE CODING FORM

```
// * DUMP CORE LOADS TO CARDS
// JOB
// DUP
*DUMPDATA   FXØ PN  ALPHA
*DUMPDATA   FXØ PN  BETA1
*DUMPDATA   FXØ PN  DELTA
*DUMPDATA   FXØ PN  GAMMA
      •
      •  (BLANK CARDS)
      •
*DUMPLET  F
// * DELETE CORE LOADS FROM FIXED AREA
// JOB
// DUP
*DELETE   M         ALPHA
*DELETE   M         BETA1
*DELETE   M         DELTA
*DELETE   M         GAMMA
*DUMPLET  F
// * RELOAD NEW SEQUENCE OF CORE LOADS
// JOB
// DUP
*STOREDATAD RD  FXØ GAMMA     ØØ1   ØØ32Ø
      •
      •  (CARD SOURCE INPUT)
      •
*STOREDATAD RD  FXØ BETA1     ØØ1   ØØ32Ø
      •
      •  (CARD SOURCE INPUT)
      •
*STOREDATAD RD  FXØ ALPHA     ØØ1   ØØ32Ø
      •
      •  (CARD SOURCE INPUT)
      •
*STOREDATAD RD  FXØ DELTA     ØØ1   ØØ32Ø
      •
      •  (CARD SOURCE INPUT)
      •
      •
      •
*DUMPLET        FX
// END
```

Figure 41. Reloading Core Loads to User Sequence

Since the source specified is the User Area, a LET search is performed. In dumping a program to NPWS, a check is made to see if there is sufficient space in the designated area; if so, a physical move operation takes place. As mentioned earlier (see Example 11) the *DUMP function converts relocatable programs from disk system format to a format of the I/O media selected. (Note that the print format to list printer is identical for both *DUMPDATA and *DUMP).

Figure 42 gives a typical sequence of control cards used.

Note that a relocatable program may also be dumped from the Temporary Area to any I/O media.

A relocatable program dumped from the User Area to punched cards may be later reloaded, if desired, to the User Area by an *STORE function.

EXAMPLE 14. MOVING A DATA FILE (OR FILES) WITHIN THE CORE LOAD AREA. This is equivalent to dumping a data file (or files) from one disk to another -- that is, the copying of process data.

Figure 43 illustrates one possibility.

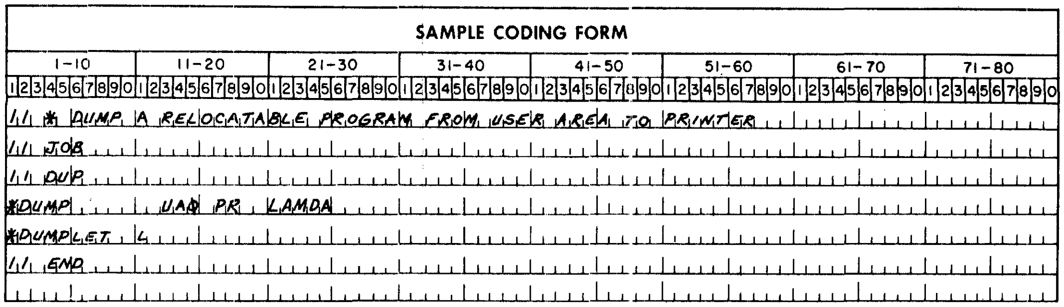
EXAMPLE 15. LOAD A PROGRAM/DATA BACK INTO THE NONPROCESS WORKING STORAGE (see Figure 44). A reloading operation implies that the program or data to be reloaded is the product of an *DUMP or *DUMPDATA function. That is, they must be in binary format (54 words per card).

Data card input decks have seventy-two columns of data, and a seven-column sequence number. Sequence columns 78-80 are assumed to be numeric (the first being 001) as punched during a *DUMPDATA function. In the reload operation, the card deck is read and stored, and a check made for consecutive sequence, modulo 1000. A sequence break is interpreted as a potential end of the deck. If the card generating the break is a *CCEND card, the *STOREDATA proceeds to store the card data directly into its destination area.

Implementation of LOCALs

An introduction to the term "LOCALs" has already been made in the introductory section: Overview of the IBM 1800 Time-Sharing Executive System. A local is classified in TSX as a subprogram or subroutine that is associated with a given core load, but not initially loaded with that core load. When a call for a local is encountered during the execution of the core load, the local is read in from the disk, overlaying the area between the end of the core load and the beginning of COMMON, unless the local is already in core. Control is then passed to the local routine.

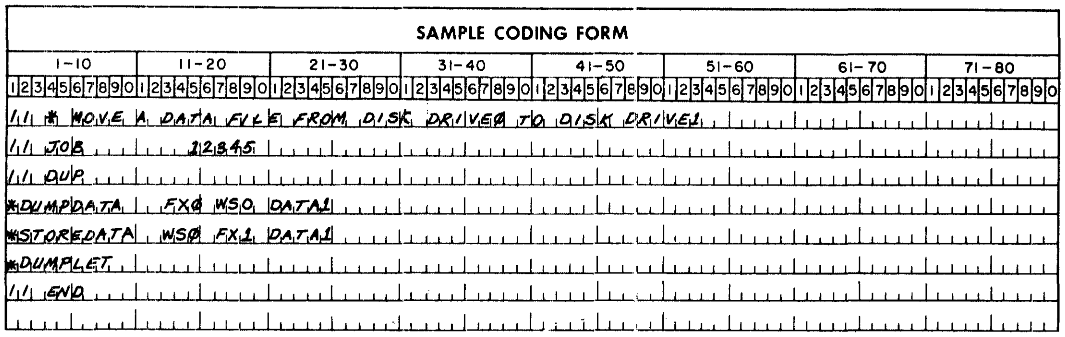
Locals may be employed as individual subprograms or groups of subprograms. In the latter case, whenever a call for a given local is encountered, the entire group of which it is a member is loaded. Subsequent calls for other locals within the same group may then be made without necessitating



SAMPLE CODING FORM

```
// * DUMP A RELOCATABLE PROGRAM FROM USER AREA TO PRINTER
// JOB
// DUP
*DUMP       UAØ PR  LAMDA
*DUMPLET  4
// END
```

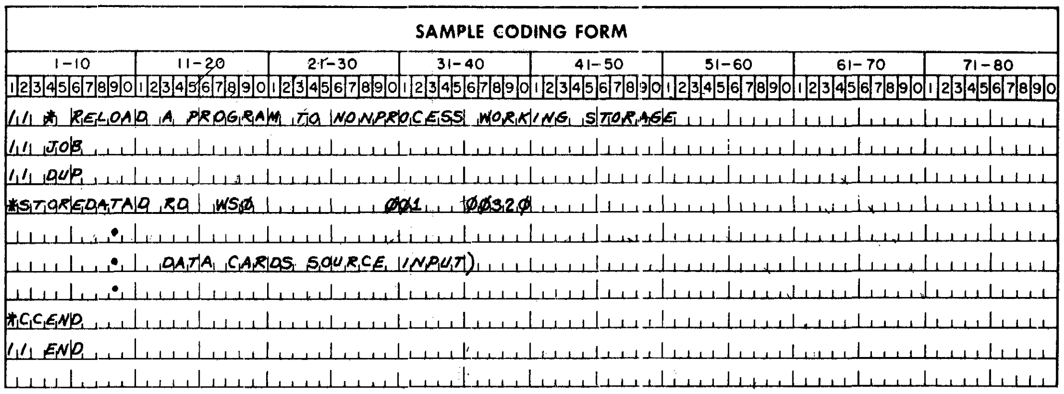
Figure 42. Dumping a Relocatable Program from the User Area



SAMPLE CODING FORM

```
// * MOVE A DATA FILE FROM DISK DRIVEØ TO DISK DRIVE1
// JOB        12345
// DUP
*DUMPDATA   FXØ WSO DATA1
*STOREDATA  WSØ FX1 DATA1
*DUMPLET
// END
```

Figure 43. Moving a Data File within the Core Load Area



SAMPLE CODING FORM

```
// * RELOAD A PROGRAM TO NONPROCESS WORKING STORAGE
// JOB
// DUP
*STOREDATAD RD  WSØ          ØØ1   ØØ32Ø
        •
        •   DATA CARDS SOURCE INPUT)
        •
*CCEND
// END
```

Figure 44. Reloading a Program to Nonprocess Working Storage

reloading from disk. Since all local groups occupy the same area in core, a call for a local in another group will involve a load from disk of the new group, overlaying the first group. At this point, another call to a local in the first group will require reloading that group from disk. Local groups or blocks are important for the following reasons: (1) specification requirements, (2) disk space utilization, and (3) disk efficiency. The area of core used is directly proportional to the size of the largest block. Data is passed to or from a local through its parameters, COMMON, or working storage.

The only difference between a local and a normally called subroutine is that a localized subroutine is not assembled as part of a core load. After the associated mainline program and all of its in-core subprograms (and their in-core subprograms) are relocated, each local subprogram block is converted to core image format and stored, sectorized, immediately following the core image core load, as shown in the schematic diagram below (Figure 45). See also Figure 3.

One advantage of the local feature is that logical subroutines can now be broken off from a large program. This means a virtual extension of core. There is no theoretical limit to the number of localized subprograms that can be implemented: the user can specify any number of locals within a group as long as the sum total of all assembled relocatable programs does not exceed the size of the Local Subroutine Area.

Communications Linkages

At object time, locals are located between the end of the main core load and COMMON. Linkage to and from locals is accomplished via a loader called FLIP (a miscellaneous subroutine within the TSX Subroutine Library) as follows.

CALLS. A call to a local consists of a BSI L X, where X is the location of a six-word entry in the Local Parameter Table (LPT) which is built by the

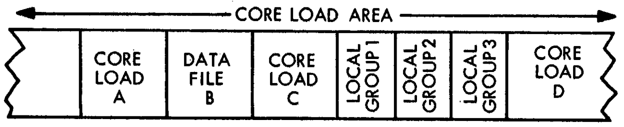


Figure 45. Showing the Relationship of Local Groups or Blocks to Associated Core Load within the Core Load Area on Disk

Core Load Builder as part of each core load in which locals are specified. The table provides the linkage between the core load and the localized subprograms via the FLIP relocatable subroutine. There is one entry in the LPT for each entry point in the specified local subroutines. Each LPT entry has the following format:

WORD	CONTENTS	MEANING
X	DC 0	A linkage word
X+1	BSI L	Long BSI to FLIP routine
X+2	FLIP	
X+3	WC	Word count of the local group with which this routine is loaded
X+4	SA	Sector address of the first sector for the local group. This address is relative to the first local sector for the core load
X+5	EP	Absolute address of the entry point when the local group is loaded

The word at X is used for the return linkage; X+1 and X+2 are executed to link to the FLIP routine which uses the word count and sector address at X+3 and X+4 to load the proper local group from disk, if required. The necessity for loading is based upon a comparison of the WC and SA words with those of the local group currently in core. The word at X+5 is the entry point of the specific local called, and control is passed to that point by the FLIP routine via a BSI I X+5 after the requisite local is loaded. Thus, the called local can return in normal fashion, and will actually return to FLIP which completes the return linkage by a BSI I X.

LIBFs. A LIBF to a local consists of a short BSI to an entry in the Variable Transfer Vector (VTV) associated with each core load. The VTV logic then executes a BSI L Y where Y is the first word of a LPT block, similar to that just described. The only difference between the two tables is that a LPT block for a LIBF function will contain a BSI L FLIP+2 instead of a BSI L FLIP. The alternate entry point allows the FLIP routine to make CALL and LIBF requests appear identical for return purposes. For a LIBF request, FLIP moves the first word of the VTV entry, set by the short BSI of the LIBF request, to the first word of the LPT block used for entry to FLIP.

Restrictions on the Use of LOCALs

Certain rules apply with respect to the constitution of locals, calling locals, and to calls made by locals. These are summarized below under legal and illegal uses.

Legal Uses

A mainline can call a local. Note that a mainline (which can be a process, interrupt, combination, or nonprocess core load) can, by definition, include any subroutines loaded with the core load. Although routines in the main core load can call locals, all such calls must be completed (that is, corresponding returns to the calling routine made) before any call on a local in a second local group can be made.

A local can call a mainline.

A local can call a skeleton subroutine.

A local can call a local provided both locals are contained in the same local group.

Illegal Uses

A local cannot call another local in a different local group.

Due to the transient nature of local routines, I/O routines cannot be designated as locals.

Conversion routines (e.g., HOLL, EBPA, PRT) cannot be designated as locals.

Interrupt servicing subroutines cannot be designated as locals.

In-skeleton subroutines cannot be used as locals.

Other Considerations

One other restriction in specifying subroutines as locals is that if a subroutine has more than one entry, i.e., EDBR, EDBRX, EDIV, and EDIVX, and more than one entry point is called, then all entry points must be indicated on the *LOCAL control card.

The user should also beware of hidden locals. If, for example, A, B, and C are subroutines, and A calls B, and B calls C, A and C should not be made locals because C would be hidden from the relocatable loader when A was prepared for loading, and on execution, local C would destroy local A. To overcome this problem, A and B, or B and C, or A, B, and C could be named as locals.

If the Local Subroutine Area includes a device I/O buffer area, no local should exit to a non-blocked (that is, non-local) subprogram until it has tested for a device routine not-busy status.

EXAMPLE 16. In certain application situations, portions of a problem program may not lend themselves to segmentation into individual core loads. In order to overcome this difficulty, by being able to contain such a program in the available machine core size, the local concept is immensely useful. The implementation of the load-on-call facility means that subroutines within the main body of a program can be called into core on demand.

The following example has been devised to illustrate this type of situation. It should not be construed as a model.

Assume a 32K system with a 16K skeleton. If all of FORTRAN I/O were used for all devices called by the nonprocess program, NCATE, core size limitations require that FORTRAN I/O be localized (see Figure 46). Since all FORTRAN I/O taken together

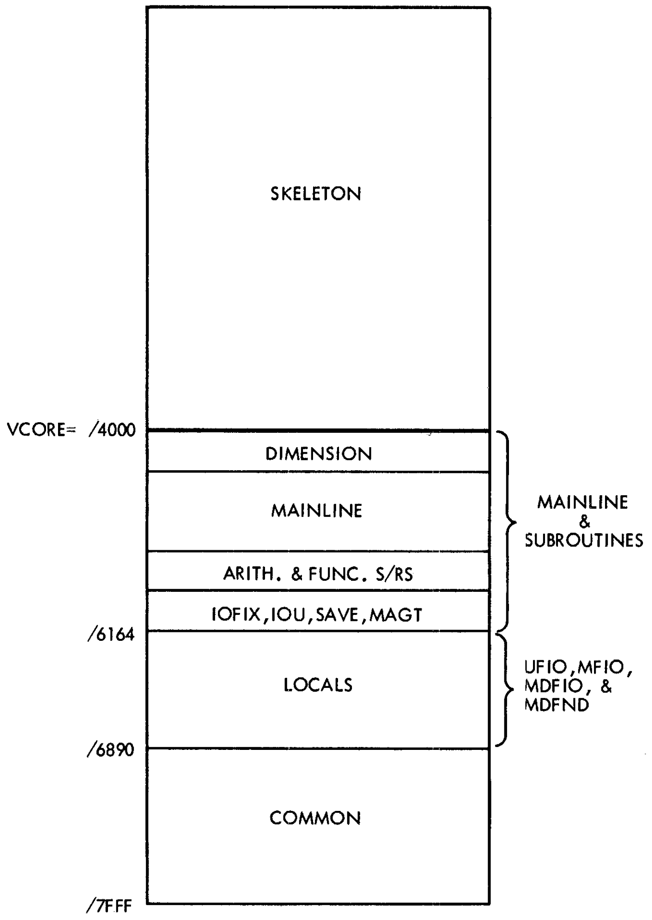


Figure 46. Illustrating the Implementation of LOCALs

would comprise approximately 3500 words, by localizing them, the largest local block will be only approximately 1600 words, and thus small enough to be accommodated within the 1830 words available.

Note that subroutines common to FORTRAN I/O (that is, called by FORTRAN I/O and not by the mainline) are automatically included in the mainline such that they may be shared. MAGT, being included in the mainline, can be referenced by either MFIO or UFIO.

Program Listing No. 4 also indicates the order of control cards acceptable to the Core Load Builder. For nonprocess programs, these must all be placed between the *STORECI (or // XEQ) control card and a *CCEND control card. Only the *RCORD control is not allowed.

PROGRAM LISTING NO. 4: EXAMPLE 16

```
// JOB
// DUP
*STOREDATAD WSO FXO FILE2       2
DUP FUNCTION COMPLETED
```

```
// JOB
// FOR NCATE
*LIST ALL
*NONPROCESS PROGRAM
*IOCS(CARD,1443 PRINTER,DISK,TAPE)
*ONE WORD INTEGERS
C
C      NONPROCESS MAINLINE--FORTRAN I/O IS LOCALIZED
C      --I/O SEGMENTATION IS ALWAYS DESIRABLE
C
C
C      LARGE DIMENSION IMPLIES SIMULATION OF EXTRA CODE
C      FOUND IN LARGE PROGRAM
C
       DIMENSION SPACE(2000),ROOM(500)
       COMMON ARRAY(2000),POINT(997),A,B,C
C
       DEFINE FILE 1(320,1,U,IFIL1)
       DEFINE FILE 2(320,1,U,IFIL2)
C
       CALL INOUT(1)
C
C      DO HIGHLY SOPHISTICATED PROGRAMMING
C
       SPACE(1) = A*B/C+A**B*ATAN(C)-B**2
       ROOM(1) = ABS(A)*ALOG(B)*EXP(C*A)/(A*COS(C)*TANH(B))
C
C      USER-WRITTEN NON I/O DUMMY PROCESSING ROUTINES
C      WHICH MIGHT BE LOCALIZED
C
C      CALL COMPT
C      CALL SURCH
C      CALL SORT
C      CALL CAMPH
C
       CALL INOUT(2)
C
       CALL EXIT
       END
VARIABLE ALLOCATIONS
 ARRAY(RC)=FFFE-F060 POINT(RC)=F05E-E896     A(RC)=E894           B(RC)=E892           C(RC)=E890       SPACE(R )=0FAA-000C
  ROOM(R )=1392-0FAC IFIL1(I )=139E      IFIL2(I )=139F

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 INOUT    FATAN    FABS     FALOG    FEXP     FCOS     FTANH    FAXB     FADD     FMPY     FDIV     FLD      FSTO     FSTOX    FSBR
 FDVR     FAXI

INTEGER CONSTANTS
     1=13A2       2=13A3

CORE REQUIREMENTS FOR NCATE
 COMMON   6000  INSKEL COMMON      0  VARIABLES   5026  PROGRAM     88


 END OF COMPILATION


NCATE
DUP FUNCTION COMPLETED
// FOR
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
*LIST ALL
C
C      DO NOTHING I/O STATEMENTS FOR ILLUSTRATIVE PURPOSES ONLY
C
       SUBROUTINE INOUT(I)
       COMMON ARRAY(2000),POINT(997),A,B,C
C
       GO TO (1,2),I
```

```
C
    1 READ(2,3) A,B,C
      FIND(2'1)
      RETURN
C
    2 READ(2'1) ARRAY
      WRITE(1'1) ARRAY
      READ(5) A,B,C
      WRITE(6) A,B,C
      END FILE 6
      BACKSPACE 5
      REWIND 6
      RETURN
C
    3 FORMAT(3F10.3)
      END
VARIABLE ALLOCATIONS
 ARRAY(RC)=FFFE-F060 POINT(RC)=F05E-E896     A(RC)=F894            B(RC)=F892            C(RC)=E890

STATEMENT ALLOCATIONS
 3     =0004  1     =0012  2     =0021

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 COMGO   URED     UWRT     UCOMP    MRED     MCOMP    MIOF     BCKSP    EOF      REWND    SUBIN    UIOF     MDRED    MDWRT    MDCOM
 MDAF    MDFND

INTEGER CONSTANTS
     2=0000       1=0001       5=0002       6=0003

CORE REQUIREMENTS FOR INOUT
 COMMON    6000  INSKEL COMMON       0  VARIABLES       0  PROGRAM      74


 END OF COMPILATION



INOUT
DUP FUNCTION COMPLETED
*STORECIL            NCATE NCATE
*FILES(2,FILE2,0)
*LOCAL(MDFIO,MDAF,MDAI,MDCOM,MDF,MDI,MDFX,MDIX,MDRED,MDWRT,MDFND)
*LOCAL(MFIO,MRED,MWRT,MCOMP,MIOAF,MIOAI,MIOF,MIOI,MIOFX,MIOIX)
*LOCAL(UFIO,URED,UWRT,UIOI,UIOF,UIOAI,UIOAF,UIOFX,UIOIX,UCOMP)
*CCEND

 CLB, BUILD NCATE

 CORE LOAD  MAP
 TYPE NAME  ARG1   ARG2

 *CDW TABLE 4002   000C
 *IBT TABLE 400E   0023
 *FIO TABLE 4031   0010
 *ETV TABLE 4041   000C
 *VTV TABLE 404D   00AE
 *PNT TABLE 40FC   0004
 *LPT TABLE 4100   00BA
 *DFT TABLE 41BA   000C
 MAIN NCATE 555E
 PNT  NCATE 40FE
 CALL FLIP  55B4
 LOCL MDFIO 4100   404D
 LOCL MDAF  4106   4050
 LOCL MDAI  410C   4053
 LOCL MDCOM 4112   4056
 LOCL MDF   4118   4059
 LOCL MDI   411E   405C
 LOCL MDFX  4124   405F
 LOCL MDIX  412A   4062
 LOCL MDRED 4130   4065
 LOCL MDWRT 4136   4068
 LOCL MDFND 413C   406B
 LOCL MFIO  4142   406E
 LOCL MRED  4148   4071
 LOCL MWRT  414E   4074
 LOCL MCOMP 4154   4077
 LOCL MIOAF 415A   407A
 LOCL MIOAI 4160   407D
 LOCL MIOF  4166   4080
 LOCL MIOI  416C   4083
 LOCL MIOFX 4172   4086
 LOCL MIOIX 4178   4089
 LOCL UFIO  417E   408C
 LOCL URED  4184   408F
 LOCL UWRT  418A   4092
 LOCL UIOI  4190   4095
 LOCL UIOF  4196   4098
 LOCL UIOAI 419C   409B
 LOCL UIOAF 41A2   409E
 LOCL UIOFX 41A8   40A1
 LOCL UIOIX 41AE   40A4
 LOCL UCOMP 41B4   40A7
 CALL INOUT 5615
 LIBF FLD   56BC   40AA
 LIBF FMPY  56D5   40AD
 LIBF FDIV  5717   40B0
 LIBF FSTO  56A2   40B3
 CALL FAXB  5781
 CALL FATAN 57C2
 LIBF FADD  588C   40B6
 LIBF FAXI  590F   40B9
 LIBF FSBR  586C   40BC
 LIBF FSTOX 5658   40BF
 CALL FABS  594E
 CALL FALOG 596A
 CALL FEXP  59FE
 CALL FCOS  5A7A
 CALL FTANH 5B0E
 LIBF FDVR  575D   40C2
 LIBF ADRCK 5B62   40C5
 CALL BT2BT 5BC6
 CALL SAVE  5BE2
 CALL IOFIX 5C46
 LIBF IOU   5C76   40C8
 CALL BT1BT 5CCE
 LIBF FLOAT 5D32   40CB
 LIBF IFIX  5D4E   40CE
 LIBF MAGT  5D7A   40D1
 LIBF SUBIN 5F98   40D4
 LIBF COMGO 5FD2   40D7
 LIBF EOF   605D   40DA
 LIBF BCKSP 6064   40DD
 LIBF REWND 6024   40E0
 LIBF FARC  60B0   40E3
 CALL FTNTR 60E4
 CALL FTRTN 60FE
 CALL FLN   597A
 LIBF FMPYX 56D0   40E6
 CALL FXPN  5A0E
 LIBF XMDS  610E   40E9
 LIBF FADDX 5886   40EC
 LIBF FSUBX 587B   40EF
 LIBF FDIVX 5712   40F2
 LIBF FLDX  56B7   40F5
 LIBF NORM  6136   40F8
 CORE       6164   072C
 COMM       6890   1770

 CLB, NCATE LD XQ

DUP FUNCTION COMPLETED
```

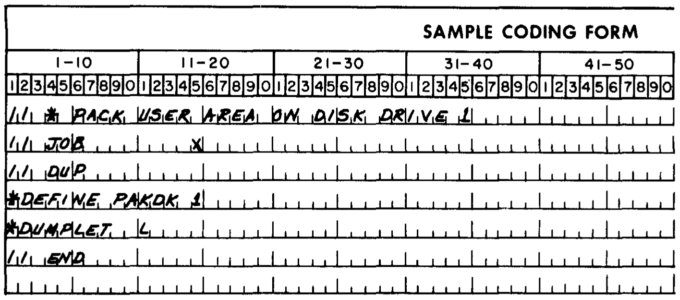
General Utility Functions

EXAMPLE 17. PACKING THE USER (RELOCATABLE PROGRAM) AREA. It has been mentioned that during a delete operation, the LET table is searched for the name of the program to be deleted and that entry replaced by a 9DUMY. The space (that is, area) previously occupied by the deleted program remains unused until an *DEFINE PAKDK operation has been performed: it then becomes available for the storage of other programs (through the *STORE function). The user is therefore advised to repack relocatable programs for optimum disk utilization at convenient intervals.

When repacking is performed, the user should ensure that a current record of disk storage exists as a safeguard against any errors which might occur while packing is in progress. The amount of time involved in this operation is directly proportional to the quantity of data moved. The sequence of control cards for a typical packing operation is given in Figure 47. Note that the *DEFINE PAKDK function serves only to pack relocatable programs on disk.

Figure 48 illustrates how various portions of the TSX subroutine library can be deleted or removed from the disk if they are not needed for a given user system. The Relocatable Subroutine Area is then packed to conserve disk space.

EXAMPLE 18. HOW TO REPRODUCE CARDS. When the input to the *STOREDATA function is in card form, this function requires the card deck to be sequenced, modulo 1000. Any form of input may exist from columns 1-72, as no conversion takes place.



SAMPLE CODING FORM

```
         1-10      11-20     21-30     31-40     41-50
1234567890123456789012345678901234567890123456789012345678901234567890
// * PACK USER AREA ON DISK DRIVE 1
// JOB         X
// DUP
*DEFINE PAKDK 1
*DUMPLET  L
// END
```

Figure 47. Repacking User Area on Disk Drive 1

This may be used to reproduce source decks prior to assembly (e.g., the TASK source deck).

EXAMPLE 19. DUMPING A LET/FLET TABLE. The *DUMPLET function is used to dump to the List Printer the contents of the LET or FLET or both tables for one or all drives specified during a particular job. The control card sequence for a LET/FLET dump is shown in Figure 50.

The format of a LET/FLET entry is summarized in Figures 51 and 52. A detailed explanation of the contents of both tables is given in the IBM 1800-Time-Sharing Executive System, Operating Procedures, Form C26-3754.

EXAMPLE 20. HOW TO CALL FOR A PROCESS CORE LOAD EXTERNALLY. Once an on-line TSX system has been built, the question remains of initializing or starting system operation. This is only possible through a cold start procedure -- Figure 53 illustrates a typical sequence of control cards for a three-drive system.

The cold start program is supplied with the IBM System and is normally resident on disk. It is read into high-addressed core storage by a two-card bootstrap (COLD START LOADER CARDS 1 and 2), and control passed to its first executable instruction. The Skeleton is then loaded to core storage, and certain mask registers in the Fixed Area are set to /FFFF thus forcing the Skeleton I/O routines to operate in a masked mode. Note that it is the user's responsibility to unmask his system, according to his configuration determined at system generation time, through his initial (that is, first) process core load.

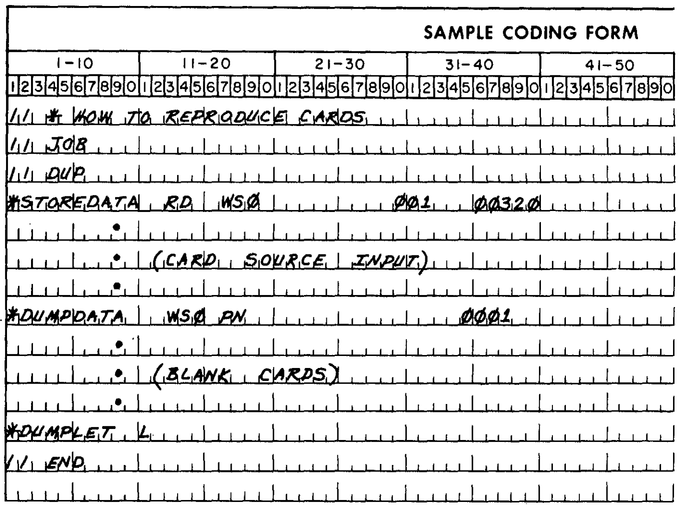
The third control card in the sequence, the COLD START name card, specifies

1. Whether or not storage protection is required (1 or 0 in column 14).
2. Whether or not a request is made for the manual entry of the time of day (1 or 0 in Column 16)
3. The logical assignments of physical disk drives on the system.
4. The name of the initial process core load.

If the storage protection option is elected, the Skeleton I/O, the System Director, the Executive Branch Table, and certain words in the Fixed Area are protected against any user violation. When the

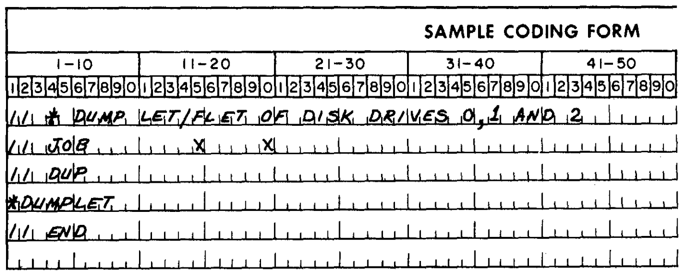
```
// JOB
// *      THE FOLLOWING SET OF TSX MONITOR CONTROL CARDS IS USED
// *      TO DELETE VARIOUS PARTS OF THE TSX SUBROUTINE LIBRARY IF THEY
// *      ARE NOT NEEDED FOR A GIVEN USER-CONFIGURATED SYSTEM
// *
// *
// *      THE FIRST DELET  ELIMINATES CARDN IF THE USER HAS ASSEMBLED
// *      TASK WITH CARDN INCLUDED--NOTE THAT FOR OFF-LINE SYSTEMS CARDN
// *      SHOULD BE INCLUDED IN TASK SINCE THIS SAVES EXECUTION TIME
// *      CORE.  I.E. CDINS EQUATED TO 1
// DUP
*DELET              CARDN
// *      IF THE USER DOES NOT HAVE  MAGNETIC TAPE ON HIS SYSTEM THE
// *      FOLLOWING DELETS APPLY
// DUP
*DELET              MAGT
*DELET              REWND
*DELET              UFIO
// *      IF THE USER DOES NOT HAVE PAPER TAPE ON HIS SYSTEM
// *      THE FOLLOWING DELETS APPLY
// DUP
*DELET              PAPTN
*DELET              PAPEB
*DELET              PAPHL
*DELET              PAPPR
// *      IF THE USER DOES NOT HAVE A PLOTTER ON HIS SYSTEM THE
// *      FOLLOWING DELETS APPLY
// DUP
*DELET              FCHAR
*DELET              SCALF
*DELET              FGRID
*DELET              FPLOT
*DELET              ECHAR
*DELET              SCALE
*DELET              EGRID
*DELET              EPLOT
*DELET              POINT
*DELET              FCHRX
*DELET              FRULE
*DELET              ECHRX
*DELET              ERULE
*DELET              XYPLT
*DELET              PLOTI
*DELET              PLOTX
// *      IF THE USER IS BUILDING AN OFF-LINE SYSTEM THE FOLLOWING
// *      DELETS APPLY--NOTE. DO NOT ASSEMBLE AND STORE THE SYSTEM
// *      DIRECTOR
// DUP
*DELET              CLEAR
*DELET              CLOCK
*DELET              COUNT
*DELET              DPART
*DELET              ENDTS
*DELET              LEVEL
*DELET              MASK
*DELET              OPMON
*DELET              QIFON
*DELET              QUEUE
*DELET              RESMK
*DELET              SAVMK
*DELET              SETCL
*DELET              TIMER
*DELET              UNMK
*DELET              UNQ
*DELET              VIAQ
*DELET              CONHX
*DELET              TRPRT
// *      IF THE USER HAS NO PROCESS I/O ON HIS SYSTEM THE FOLLOWING
// *      DELETS APPLY
// DUP
*DELET              AIPTN
*DELET              AISQN
*DELET              AIRN
*DELET              ANINT
*DELET              DINP
*DELET              DIEXT
*DELET              DICMP
*DELET              DAOP
*DELET              IOPE
*DELET              XSAVE
*DELET              GAGED
*DELET              AIP
*DELET              AIS
*DELET              AIR
*DELET              CS
*DELET              CSC
*DELET              CSX
*DELET              DAC
*DELET              QZERQ
*DELET              QZ010
// JOB
// *      THE SUBROUTINE AREA WILL NOW BE PACKED TO CONSERVE DISK SPACE
// DUP
*DEFINE PAKDK 0
// JOB
// END OF EXTRA SUBROUTINE DELETS
```

Figure 48. Repacking the Relocatable Subroutine Area Following a Removal of Various Portions of the TSX Subroutine Library



```
SAMPLE CODING FORM
   1-10      11-20     21-30     31-40     41-50
1234567890123456789012345678901234567890123456789 0
// * HOW TO REPRODUCE CARDS
// JOB
// DUP
*STOREDATA  RD  WSØ          ØØ1     ØØ320
         .
         .  (CARD SOURCE INPUT)
         .
*DUMPDATA   WSØ PN               ØØØ1
         .
         .  (BLANK CARDS)
         .
*DUMPLET  L
// END
```

Figure 49. Reproduction of Cards



```
SAMPLE CODING FORM
   1-10      11-20     21-30     31-40     41-50
12345678901234567890123456789012345678901234567890
// * DUMP LET/FLET OF DISK DRIVES 0,1 AND 2
// JOB         X    X
// DUP
*DUMPLET
// END
```

Figure 50. Dump LET/FLET of Disk Drives 0, 1 and 2

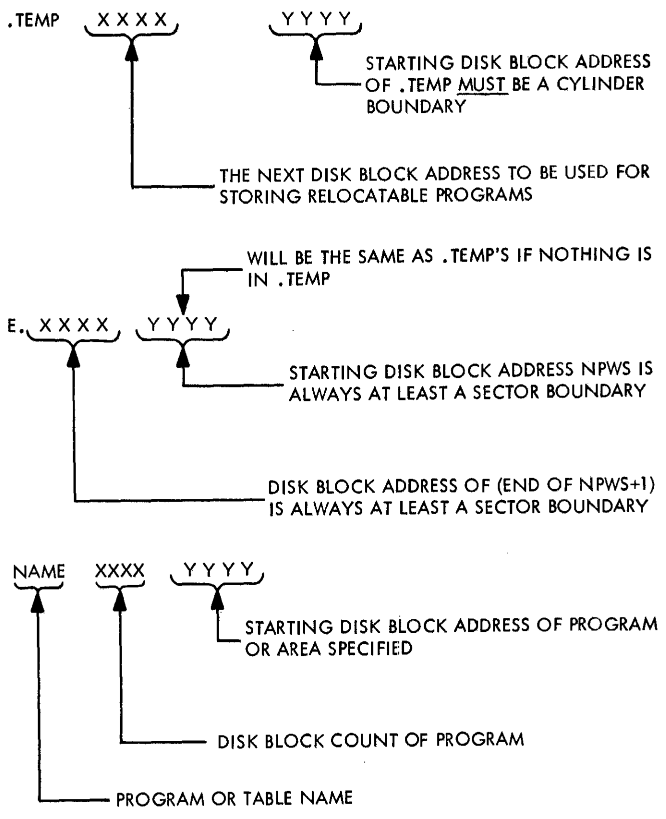

.TEMP
XXXX
YYYY
STARTING DISK BLOCK ADDRESS OF .TEMP MUST BE A CYLINDER BOUNDARY
THE NEXT DISK BLOCK ADDRESS TO BE USED FOR STORING RELOCATABLE PROGRAMS
WILL BE THE SAME AS .TEMP'S IF NOTHING IS IN .TEMP
E.
XXXX
YYYY
STARTING DISK BLOCK ADDRESS NPWS IS ALWAYS AT LEAST A SECTOR BOUNDARY
DISK BLOCK ADDRESS OF (END OF NPWS+1) IS ALWAYS AT LEAST A SECTOR BOUNDARY
NAME
XXXX
YYYY
STARTING DISK BLOCK ADDRESS OF PROGRAM OR AREA SPECIFIED
DISK BLOCK COUNT OF PROGRAM
PROGRAM OR TABLE NAME

Figure 51. LET Entries

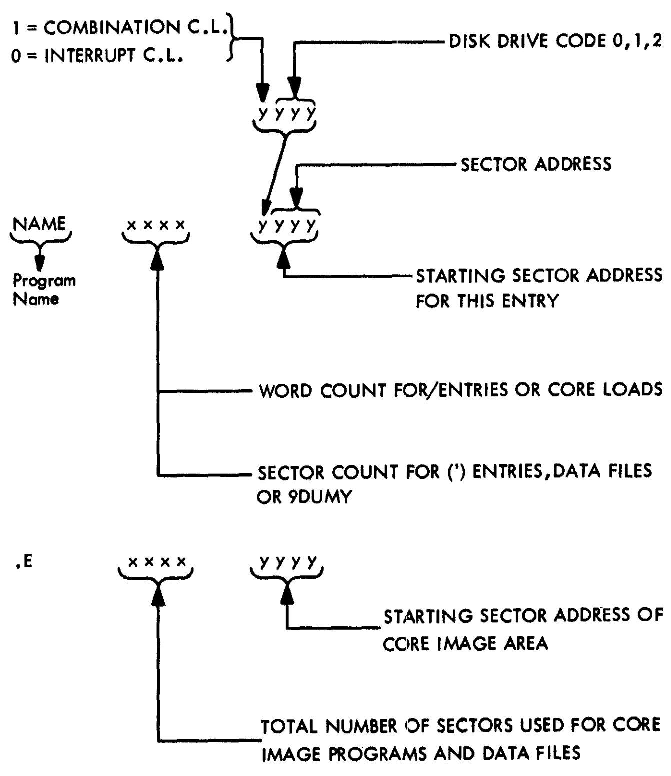

1 = COMBINATION C.L.
0 = INTERRUPT C.L.
DISK DRIVE CODE 0,1,2
yyyy
SECTOR ADDRESS
NAME
Program Name
xxxx
yyyy
STARTING SECTOR ADDRESS FOR THIS ENTRY
WORD COUNT FOR/ENTRIES OR CORE LOADS
SECTOR COUNT FOR (') ENTRIES, DATA FILES OR 9DUMY
.E
xxxx
yyyy
STARTING SECTOR ADDRESS OF CORE IMAGE AREA
TOTAL NUMBER OF SECTORS USED FOR CORE IMAGE PROGRAMS AND DATA FILES

Figure 52. FLET Entries

SAMPLE CODING FORM

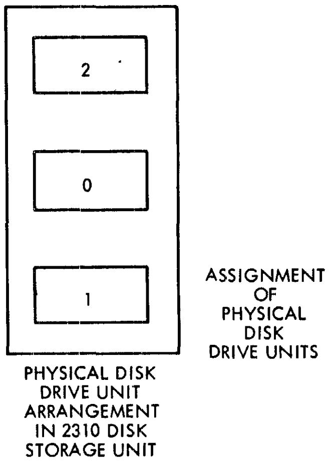
```
// * PREPARE SYSTEM FOR ON-LINE OPERATION
// * COLD START LOADER CARD 1 SUPPLIED WITH IBM SYSTEM
// * COLD START LOADER CARD 2 SUPPLIED WITH IBM SYSTEM
*CLDST START 1 1 Ø 1 2 START PROCESS CORE LOAD NAME CARD FOR COLD START
```

Figure 53. Cold Start for an On-Line System

clock option is selected, the user manually enters the time of day in decimal hours and minutes (switches 0-7 and 8-15 respectively): when CONSOLE START is depressed, this is converted into hexadecimal hours and thousands of hours (see also System Design Considerations: System Director).

The assignment of physical disk drive units in a multi-disk system is based on a logical scheme to give maximum flexibility, as shown in Figure 54.

Note that the physical arrangement of the disk drive units (up to three) in a 2310 Disk Storage Unit is fixed in the sequence: disk drive 2, disk drive 0, and disk drive 1. Columns 18, 20, and 22 on the COLD START name card always designate a logical number sequence: 0, 1, and 2 in that order. These columns are used at cold start time to establish a relationship between a physical disk drive (either 2, 0, or 1) and its equivalent logical reference. For example, a 1 punched in column 18 means that a program that references logical 0 will refer to the physical drive (disk drive 1) which was assigned at cold start time to that logical number (0). In Figure 53, physical disk drives 0, 1, and 2 have been assigned to logical 0, 1, and 2 respectively. One of the advantages of this flexibility in assigning physical disk drives in a multi-disk system is backup capability.


2
0
1
PHYSICAL DISK DRIVE UNIT ARRANGEMENT IN 2310 DISK STORAGE UNIT
ASSIGNMENT OF PHYSICAL DISK DRIVE UNITS

COLD START NAME CARD		
COL. 18 (LOGICAL NO. 0)	COL. 20 (LOGICAL NO. 1)	COL. 22 (LOGICAL NO. 2)
0	1	2
1	0	2
2	1	0
2	0	1
1	2	0
0	2	1

Figure 54. Relationship of Physical Disk Drive Units to Logical Number

EXAMPLE 21. HOW TO INITIATE A NONPROCESS MONITOR OPERATION. In an on-line system:

- CALL SHARE from a mainline program only, or
- CALL VIAQ (when the queue is empty). This forces a CALL SHARE.

CALL SHARE is deliberately used when time-sharing is desired at specific times and for specific durations. The amount of time is specified by the I parameter, and is variable depending upon the length of time the user wishes to be away from his process on the mainline level. This time is set in the programmed timer run under Timer C. Time-sharing is terminated when the timer returns to zero or is, alternatively, set to zero by a CALL ENDTS statement (see also System Design Considerations: System Director).

A CALL VIAQ when the Queue Table is empty forces a CALL SHARE statement: the time used in the CALL SHARE is the value set by TISHA (see System Design Considerations: System Director; also Use of Time-Sharing).

In an off-line system, Nonprocess Monitor operation may be initiated by

1. A COLD START TASK procedure, or
2. Loading a TASK object deck to core with a four-card High Core TASK Loader.

The COLD START TASK procedure is identical to the on-line COLD START PROCEDURE (see Example 20), except that the TASK operating system is now read into core storage (instead of the System Skeleton). A typical sequence of control cards is shown below (Figure 55).

An alternative method of starting an off-line system is to load a TASK object deck (previously assembled to user specification) to core-storage with a four-card bootstrap loader (High Core TASK Loader). The procedure is summarized below (see also IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754).

- Clear core. The 16 data switches may be set off, or to some predetermined value. Depress CLEAR CORE and START buttons simultaneously.
- Depress STOP button
- Reset registers to zero. Depress RESET button.
- Ready Card Reader. Depress PROGRAM LOAD on reader.
- Set Sense Switch 7 up. Depress CONSOLE INTERRUPT.
- Depress START button (on response to sense switches).

EXAMPLE 22. HOW TO TERMINATE A NONPROCESS MONITOR OPERATION (OFF-LINE SYSTEM UNDER TASK CONTROL). Two methods are possible:

1. Set Sense Switch 7 up
   Depress CONSOLE INTERRUPT

This immediately aborts the current job being processed, and proceeds to next stacked job.

2. Whenever the Card Reader is empty, the Nonprocess Supervisor will indicate this situation by printing the following message:

   N04 READER READY

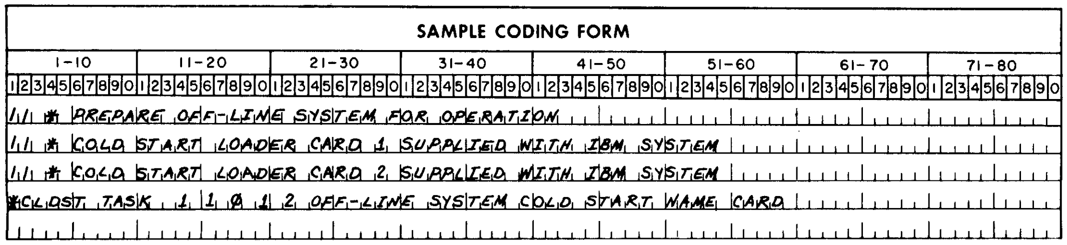
   Place next stacked job deck on hopper. Ready reader. Depress START.

EXAMPLE 23. PREPARING A GUARD (DUMMY) INTERRUPT CORE LOAD. If an interrupt occurs on a level designated as "out-of-core" and there is no interrupt or combination core load associated with it, the interrupt will be recorded automatically. To prevent this, it is good practice to provide a guard or dummy interrupt core load to service all interrupts or all assigned out-of-core interrupt levels until each interrupt has its final interrupt core load built and stored on disk. The substitute core load should give some indication (such as a message) that the interrupt has occurred.

In the example (Figure 56), levels 8, 9, 10, and 11 were defined as "out-of-core" interrupt levels by the System Director equate cards ICLL1 and ICLL2. The relocatable main program is identified by GUARD located in the temporary portion of LET. Its entry address is 5. The interrupt core load is also identified by GUARD but is in FLET. The DICLE statement specifies that GUARD is entered in the ICL Table for each bit position on each level assigned. When the named program is later deleted and replaced by another program, all of the ICLT entries will be replaced.

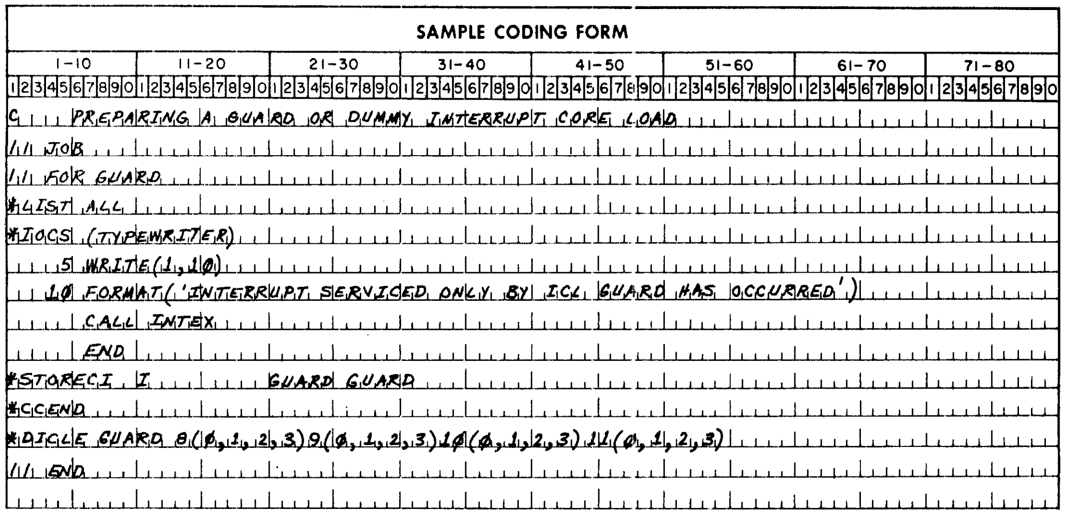
EXAMPLE 24. USE OF THE CONSOLE INTERRUPT. The Console Interrupt is used by the system and may also be used by the user.

The system uses the Console Interrupt with sense switch 7 on either to abort a nonprocess job or to


SAMPLE CODING FORM

```
//  * PREPARE OFF-LINE SYSTEM FOR OPERATION
//  * COLD START LOADER CARD 1 SUPPLIED WITH IBM SYSTEM
//  * COLD START LOADER CARD 2 SUPPLIED WITH IBM SYSTEM
*CLDST TASK  1 1 Ø 1 2 OFF-LINE SYSTEM COLD START NAME CARD
```

Figure 55. Cold Start for an Off-Line System



SAMPLE CODING FORM

```
C     PREPARING A GUARD OR DUMMY INTERRUPT CORE LOAD
// JOB
// FOR GUARD
*LIST ALL
*IOCS (TYPEWRITER)
    5 WRITE(1,1Ø)
   1Ø FORMAT('INTERRUPT SERVICED ONLY BY ICL GUARD HAS OCCURRED')
      CALL INTEX
      END
*STORECI  I         GUARD GUARD
*CCEND
*DICLE GUARD 8(Ø,1,2,3)9(Ø,1,2,3)1Ø(Ø,1,2,3)11(Ø,1,2,3)
// END
```

Figure 56. Preparing a GUARD or Dummy Interrupt Core Load

commence Nonprocess Monitor action. This operation is required after a // JOB, // END OF ALL JOBS combination.

A block diagram of the generalized logic flow is given in Figure 57.

The user may have an interrupt program executed on a chosen level by depressing the Console Interrupt button with sense switch 7 off. The level is assigned by the user on the equate card CONTA at TASK assembly time.

The servicing routine or core load is written by the user and handled in the same way as a programmed interrupt servicing routine, with appropriate LLBB designations. It is through the programmed interrupt servicing routine that the Console sense/data switches are interrogated and which, in turn, direct this routine to the course of action desired.

One of the functions the servicing routine or core load must perform is to queue up a mainline core load which will notify the Customer Engineer when he can depress the C.E. Interrupt button for the removal or addition of I/O devices from the system; and also to print out error counters where hardware malfunctions have been recorded.

EXAMPLE 25. PREPARING A MAINLINE CORE LOAD TO PERMIT PROLONGED EXECUTION OF THE NONPROCESS MONITOR FOR THE DEBUGGING OF PROCESS PROGRAMS. The example illustrates the use of the CALL SHARE statement which will continue to provide time for Nonprocess Monitor operation when the increment I for time-sharing has expired and the mainline core load is reentered.

This mainline core load will be specified in the COLD START procedure when only nonprocess work is to be accomplished.

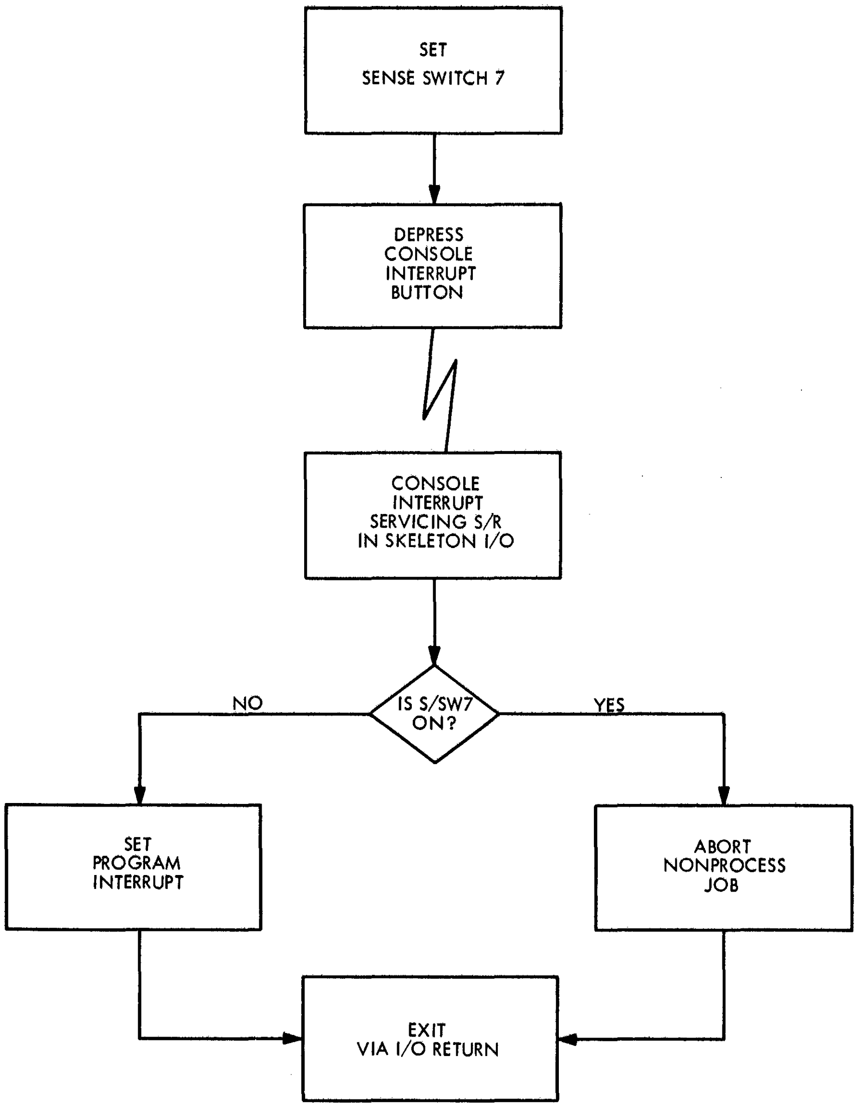
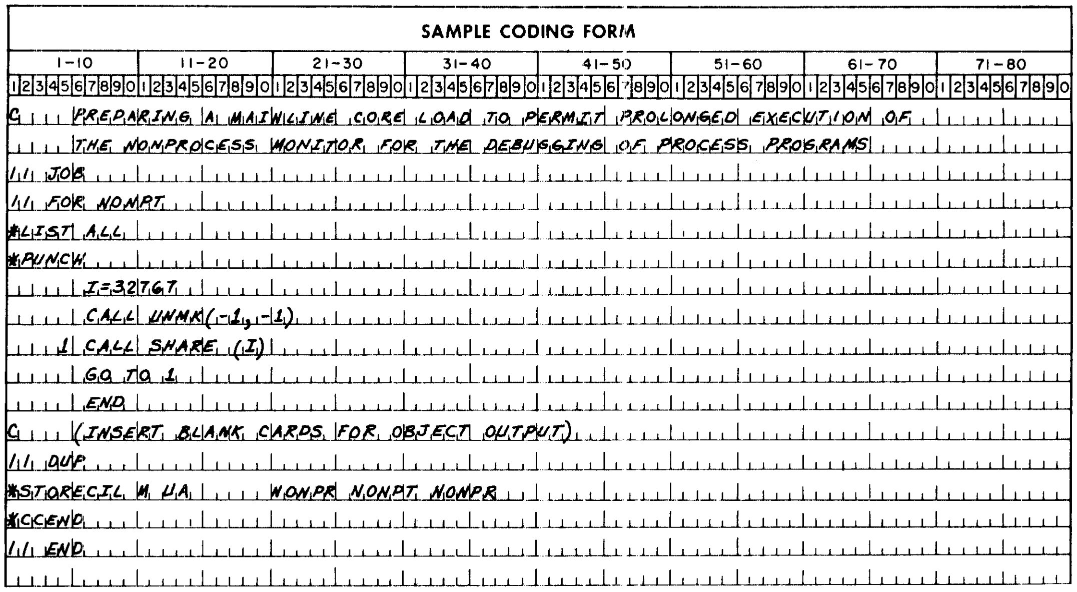


Figure 57. Illustrating Logic of Console Interrupt



SAMPLE CODING FORM

```
C     PREPARING A MAINLINE CORE LOAD TO PERMIT PROLONGED EXECUTION OF
      THE NONPROCESS MONITOR FOR THE DEBUGGING OF PROCESS PROGRAMS
// JOB
// FOR NONPT
*LIST ALL
*PUNCW
      I=32767
      CALL UNMK(-1,-1)
    1 CALL SHARE (I)
      GO TO 1
      END
C     (INSERT BLANK CARDS FOR OBJECT OUTPUT)
// DUP
*STORECIL M UA      WONPR NONPT NONPR
*CCEND
// END
```

Figure 58. Illustrating Perpetual Time-Shared Nonprocess Monitor Operation

In industrial control systems, individual user installation requirements may vary from installation to installation either in the hardware itself or in dissimilarities inherent in the application. These differences may take the form of:

- Different processes
- Special process I/O hardware
- Different input-output configurations
- Different core storage sizes
- Response time
- Throughput
- Priority considerations

This means that each installation must be defined or tailored to the specific system function requirements and input-output-configuration of that installation. The tailoring function is defined as system generation which provides the facilities for the creation and maintenance of a monitored system composed of IBM and user-written programs and subroutines. The end product of system generation is a disk-resident operating system which is custom built to provide an efficient Executive System for a specific machine environment.

In the IBM 1800 Time-Sharing Executive System, the builder or "tailor" is a stand-alone monitor program called the Temporary Assembled Skeleton (TASK). TASK permits a system to be constructed on one or more disk cartridges from absolute and relocatable program decks which contain the executable phases and the relocatable programs the installation elects to include in its system. Furthermore, the installation may modify the IBM-supplied configuration, delete functions not required by the installation and add installation-created functions and programs. The modular design and availability of many features and attachable units make possible numerous IBM 1800 configurations tailored to individual application requirements.

System Generation

As noted above, System Generation is the process of preparing a specially-tailored operating system to match the machine configuration and operating system options selected by the user. In general, two types of systems may be generated:

1. An on-line system,
2. An off-line system.

On-line System

An on-line system is one that responds continuously to the demands of the real-time world. For example, in industrial process control systems, a number of rapidly changing variables must be monitored, analyzed, and controlled at all times to produce an optimum result. A TSX on-line system implies a real-time operating system in which user-written programs continuously monitor and control a process operation under the command of an executive program (the System Director). The executive provides a means of supervising the use of input-output data and communications channels, evaluating and interpreting data, transmitting and storing information and programs, detecting and correcting errors, and interlacing time-sharing functions. It also controls the system's response to various optional requests, giving priority to emergency demands and postponing low-priority requests that may require considerable time to perform. Emergency actions can be scheduled at frequent intervals. This immediate response is secured through the medium of a powerful and flexible priority interrupt system.

In the on-line mode, the executive also permits the system to be time-shared (when free time is available) by the controlled process and unrelated nonprocess functions. This means that nonprocess programs may be assembled, compiled, simulated, and debugged without interfering with the on-line process. It is the rule rather than the exception that process control programs are subject to change, and it is a definite advantage to be able to implement changes at the installation without taking the system off-line.

Off-line System

An off-line system is completely unrelated to the real-time world, its main purpose being the handling of sequential job operations under the control of a monitor system. A TSX off-line system, by definition, constitutes a stack-job nonprocess monitor system which functions under the direction of TASK. In this mode, nonprocess operations such as assemblies, compilations, disk utility operations, and execution of user-written programs may be performed.

Since TASK core size is considerably less than System Skeleton core size, core storage requirements are less demanding for the off-line system. Also, as the various disk save areas are not now required, disk space is conserved. For those users who do not plan to utilize time-sharing, a nonprocess monitor system working under TASK gives the ability to build coreloads for an on-line system. It is from a non-process monitor system that an on-line system is ultimately constructed.

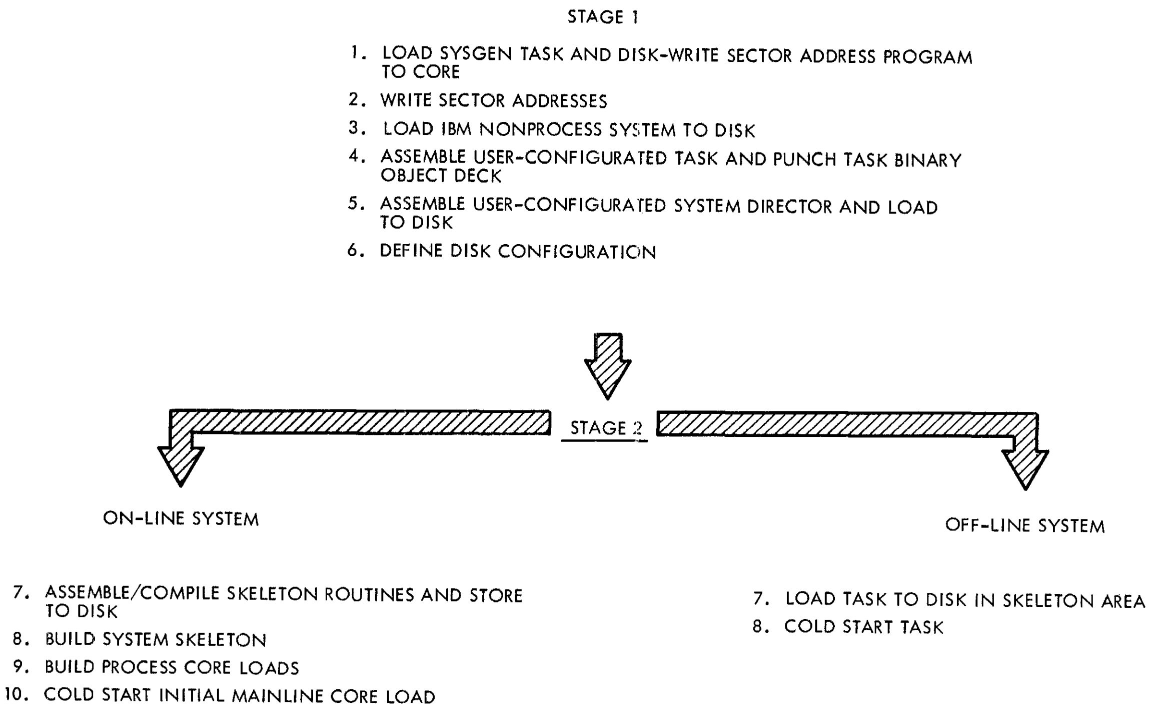
Summary of System Generation Procedures

The process of generating an on-line or off-line TSX system can be accomplished in two stages, as illustrated in Figure 59.

In stage 1, the "starter" system (SYSGEN TASK) initiates the system generation process and directs the disk write address function, loading of the IBM Nonprocess System, TASK and System Director assembly, and definition of the disk system configuration. This set of procedures is common to both types of systems.

In stage 2, the user exercises the option, depending on his application requirements, of building an on-line or off-line system cartridge.

This method of generating a completely new executive system makes use of standard system components such as TASK, the System Loader, the IBM Nonprocess System, the System Director, etc. Some of the important concepts and considerations involved in using these components will now be covered in some detail.


STAGE 1

1. LOAD SYSGEN TASK AND DISK-WRITE SECTOR ADDRESS PROGRAM TO CORE
2. WRITE SECTOR ADDRESSES
3. LOAD IBM NONPROCESS SYSTEM TO DISK
4. ASSEMBLE USER-CONFIGURATED TASK AND PUNCH TASK BINARY OBJECT DECK
5. ASSEMBLE USER-CONFIGURATED SYSTEM DIRECTOR AND LOAD TO DISK
6. DEFINE DISK CONFIGURATION

STAGE 2

ON-LINE SYSTEM

7. ASSEMBLE/COMPILE SKELETON ROUTINES AND STORE TO DISK
8. BUILD SYSTEM SKELETON
9. BUILD PROCESS CORE LOADS
10. COLD START INITIAL MAINLINE CORE LOAD

OFF-LINE SYSTEM

7. LOAD TASK TO DISK IN SKELETON AREA
8. COLD START TASK

Figure 59. System Generation Overview

For details of step-by-step system generation operational procedures, the user is referred to the System Reference Library, IBM 1800 Time-Sharing Executive System, Operational Procedures, Form C26-3754.

TEMPORARY ASSEMBLED SKELETON (TASK)

TASK EQUATE CARDS

Before TASK can be used to tailor a TSX system, like the System Director, TASK itself has to be assembled from a source deck. To do this, two groups (Groups 1 and 2) of equate cards must be physically placed in the TASK source deck to define the particular system. The relationship of the equate cards to the source deck is illustrated in Figure 60.

The size of the assembled TASK is directly proportional to the number of TASK functions the user elects to include in his system. For example, if he decides to include the complete TASK utility package which will assist him to debug his programs prior to a skeleton build, he equates TRORG to 1. If he decides to include CARDN in the Skeleton I/O, he equates CDINS to 1. If he decides to make use of the 1053/1816 backup capability, he equates the BD1-BD8 cards accordingly to identify the backup printer(s) assigned.

Like the System Director, TASK can be assembled with extreme flexibility so that no core is wasted by selecting any of the numerous options available. Furthermore, portions of the package can be deleted. The user thus elects a configuration that best matches the functions required. This is illustrated by the example given in Figure 61

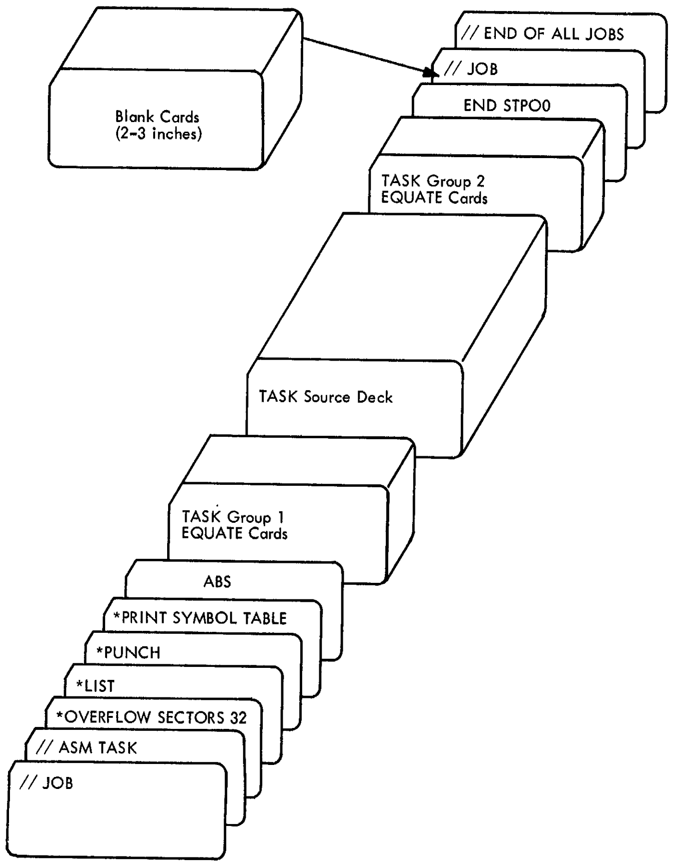


Figure 60. TASK Source Deck and TASK Equate Cards

```
*IBM 1800 TSX SAMPLE SYSTEM TASK EQUATE CARDS
CORSZ EQU     32       OBJECT SIZE IS 32K
COMSZ EQU     01000    INSKEL COMMON SIZE IS 1000 WORDS
DORG1 EQU     1        NOT A ONE-DRIVE SYSTEM
DORG2 EQU     0        THIS IS A TWO-DRIVE SYSTEM
PRIL0 EQU     01       INTERRUPT LEVEL OF DRIVE ZERO IS 01
PRIL1 EQU     02       INTERRUPT LEVEL OF DRIVE ONE IS 02
PRIL2 EQU     00       THERE IS NO DRIVE TWO
TORG  EQU     1        SYSTEM HAS 1816 KEYBOARD
TORG1 EQU     1        MORE THAN ONE 1053/1816 GROUP 1
TORG2 EQU     1        MORE THAN TWO 1053/1816 GROUP 2
TORG3 EQU     0        SYSTEM HAS THREE 1053/1816 GROUP 1
TORG4 EQU     1        ONE 1816 KEYBOARD GROUP 1
TORG5 EQU     0        NO 1816/1053 GROUP 2
TORG6 EQU     1        OTHER THAN ONE 1053/1816 GROUP 2
TORG7 EQU     1        OTHER THAN TWO 1053/1816 GROUP 2
TORG8 EQU     1        OTHER THAN THREE 1053/1816 GROUP 2
TORG9 EQU     0        NO 1816 KEYBOARD GROUP 2
TORGN EQU     1        SYSTEM HAS 1816/1053 PRINTERS
BZ1   EQU     090      MESS UNIT SIZE FOR 1053-1 GROUP 1
BZ2   EQU     090      MESS UNIT SIZE FOR 1053-2 GROUP 1
BZ3   EQU     090      MESS UNIT SIZE FOR 1053-3 GROUP 1
BZ4   EQU     090      MESS UNIT SIZE FOR 1053-4 GROUP 1
BZ5   EQU     090      MESS UNIT SIZE FOR 1053-1 GROUP 2
BZ6   EQU     090      MESS UNIT SIZE FOR 1053-2 GROUP 2
BZ7   EQU     090      MESS UNIT SIZE FOR 1053-3 GROUP 2
BZ8   EQU     090      MESS UNIT SIZE FOR 1053-4 GROUP 2
NOCYL EQU     20       20 CYLINDERS FOR MESS BUFF
NUMBE EQU     16       16 NONPROCESS MESS BUFF SECTORS
NOBUF EQU     1        DISK MESSAGE BUFFERING
TYPL1 EQU     04       1053/1816 GROUP 1 INT LEVEL 04
TYPL2 EQU     00       1053/1816 GROUP 2 INT LEVEL 00
INTKY EQU     15       USER KYBD REQ RTN INT LEVEL 15
PORG  EQU     1        1443 PRINTER ON SYSTEM
LVPR1 EQU     05       1443 PRINTER INTERRUPT LEVEL 05
LORG1 EQU     1        LIST PRINTER IS 1443
SORG1 EQU     1        SYSTEM PRINTER IS 1443
SLORG EQU     1        CARD INEFFECTIVE SEE LORG1/SORG1
ECPT1 EQU     0        EAC PRINTER IS A 1053
ECPT2 EQU     07       EAC COMBINATION EQUATE VALUE IS 7
ECPT3 EQU     0        EAC PRINTER IS A 1053 GROUP 1
CRDNO EQU     0        ONE 1442 ON SYSTEM
CDINS EQU     1        CARDN IS IN SKELETON I/O
ORLP1 EQU     1        OVERLAP ON ANALOG I/P BASIC
ORLP2 EQU     1        OVERLAP ON ANALOG I/P EXPANDER
PTSKP EQU     1        LOOPS UNTIL READY IN NONPROCESS MODE
NULEV EQU     16       15 INT LEVELS IN SYSTEM
MKLEV EQU     1        15 OR MORE INT LEVELS
CONTA EQU     14       LEVEL OF USER CONSOLE INT RTN IS 14
PRICS EQU     0        STANDARD PRECISION IS USED
TRORG EQU     1        TASK UTILITY PACKAGE INCLUDED
TA01  EQU     1        FULL TRACE INCLUDED
TA02  EQU     1        CHECK STOP TRACE INCLUDED
TA03  EQU     1        DISK DUMP INCLUDED
ONLIN EQU     1        ALL TASK FUNCTIONS ARE USED
BDT1  EQU     DT2      1053-2 GROUP 1 BACK-UP UNIT
BDT2  EQU     DT3      1053-3 GROUP 1 BACK-UP UNIT
BDT3  EQU     DT1      1053-1 GROUP 1 BACK-UP UNIT
BDT4  EQU     DT1      1053-1 GROUP 1 BACK-UP UNIT
BDT5  EQU     DT1      1053-1 GROUP 1 BACK-UP UNIT
BDT6  EQU     DT1      1053-1 GROUP 1 BACK-UP UNIT
```

Figure 61. A Set of TASK Equate Cards for the TSX Sample System (see Programming Techniques)

which depicts a set of TASK equate cards chosen for the TSX Sample System described in Programming Techniques. The significance of each of the 60 cards is clearly denoted. The majority of these cards are self-explanatory; a few, however, call for some explanation. These include the following:

- NOBUF
- BZ1-8
- NOCYL
- NUMBE
- INTKY
- CONTA
- ECPT2
- CDINS
- PRICS
- ONLIN
- COMSZ

NOBUF. This label indicates whether or not the buffering of messages to disk is required. It should be equated to zero if the user

1. Has a 16K - 32K system where very few messages on all typewriters are printed
2. System is restricted in skeleton core space.

NOBUF should be equated to 1 if the user has a 16K - 32K system, and has adequate core space (about 300 words) for the buffering feature in TYPEN.

BZ1-8 (Message Unit Size). If the user has adequate core space, makes efficient use of disk space, or if he plans to print long messages, the message unit size should be large. If, however, core space is restricted, but there is sufficient disk space for a number of sectors for the buffering of messages, the message unit size should be small -- that is, of the order of 20 - 40 words.

In general, increasing the buffer size results in a more efficient use of disk space and a corresponding less effective utilization of core storage (see Buffering of Messages to Disk).

NOCYL. This should be basically equal to the largest possible message capacity in disk cylinders at any point in time.

For example, assume that all messages for a specific system are each less than 40 words long, and that the message unit size for all 1053 printers is 40 words. Then, if in any 10 minute period, the user calls for 80 messages to be printed, NOCYL should be equated to 10. That is, an 80-sector buffer is reserved. Note that 8 words of core storage are reserved for every increment of NOCYL (see Buffering of Messages to Disk).

NUMBE. This specifies what percentage of disk message buffer can be in use by a nonprocess program in any given moment of time. It can never exceed NOCYL X 8.

INTKY. This specifies the interrupt level on which the user will service the 1816 device as the result of a keyboard request interrupt. If the user plans to use an out-of-core interrupt servicing program for this purpose, INTKY must be equated to an interrupt level lower in priority than all other I/O interrupt levels.

CONTA. The user must specify the level to be program-interrupted for the servicing of the Console Interrupt.

The interrupt servicing routine would then interrogate the sense/data switches to determine the course of action required by the interrupt.

One of the uses of this routine is to queue up a mainline core load that will enable the Customer Engineer to utilize the C. E. Interrupt facility. (See INTKY; also Examples of Nonprocess Monitor Usage -- Example 24.)

ECPT2. If two 1053 Printers form part of the user's valid system, he should always define (at least) these two printers as EAC printers for backup purposes.

CDINS. For off-line systems, this should always be equated to 1. Note that this saves about 300 words of variable core. For on-line systems, this should be zero unless the user plans to include in the skeleton a subroutine which calls the 1442 card reader.

PRICS. The user must anticipate what type of arithmetic precision is required in his process programs. He should remember that once this is defined, subroutines used by process programs are assembled with this same precision.

ONLIN. If the user plans to operate (only) an off-line disk monitor system, this should be equated to zero. This gives the user 600 more words of variable core.

COMSZ. This equate card specifies the size of INSKEL COMMON. In an off-line system, this card has no effect since INSKEL COMMON is only present in an on-line system.

BUFFERING OF MESSAGES TO DISK

Efficient I/O handling is the most important single factor in the effective utilization of processor time. Input-output devices, being slow compared to the internal speed of the processor, must be programmed to overlap their operation with mainline computations whenever possible to

1. Greatly increase efficiency of I/O operations
2. Provide more throughput of data.

Consider the following situation. The incore 1053 Printer buffer (whose size is determined by the TASK equate cards BZ1-8) contained within the D.P. I/O subroutine TYPEN is full, and the printer is in the process of writing a message. If disk buffering were provided, the next message called would be temporarily stored on disk, and later returned to core when the current message is completed. This means that the processor-controller is not locked up and waiting for the input-output operation to be completed, and is thus able to continue with its processing.

The significance of disk buffering is that queueing of output messages or information can now be easily accomplished without putting excessive loads on core size or disk access capabilities of the system.

Without disk buffering, the system becomes printer-limited, and might deteriorate into a 15 character per second system.

We thus see that the buffering of messages to disk is important for two reasons:

1. It maximizes processor time. That is, it allows computing to continue after a call to the printer is given.
2. It frees the user from having to optimize his message requests, thus permitting more effective use of the device.

The interrelationship between disk message buffering and total skeleton core requirements can be shown by the following example.

Without buffering:	Assume four 1053 Printers
	Minimum message unit size = 81 words
	Total core = 324 words
With buffering:	Assume four 1053 Printers
	Minimum message unit size = 20 words
	Add additional portion of TYPEN = 300 words
	Total core = 380 words

It is seen, in this example, that the user obtains all the advantages of buffering at the small sacrifice of 56 words.

Message Unit Size

The user must define the message unit size for the 1053 Printer(s) attached to his 1800 TSX System at TASK assembly time. The printers may belong to Group 1 or Group 2 on the condition that the maximum number of 1053 Printers used does not exceed 8.

Message unit size is defined as somewhat larger than the average size of the message or information to be printed out. This may be within the range of 20-319 words which is dictated by the minimum and maximum core sizes that may be specified for a message buffer. In practice, an optimum size may fall between 40 and 80 words (80 to 160 characters).

Definition of the message unit size is also dependent on whether messages to the 1053 Printer are to be buffered.

If non-buffering is employed, the message must never be greater than that defined for a message unit. If the user plans to print out long messages or a large number of messages; has adequate core storage, and makes efficient use of disk space, the message unit size should be large. Assuming FORTRAN compilation is planned, the message unit size should be at least 81 words (162 characters).

If buffering is preferred (because the user is pressed for core storage, but has enough disk space for a number of sectors for the buffering of messages), the size of a message can be any length; that is, greater than the size of the message unit. The message unit size can now be defined as small as 20 words (40 characters).

In general, a large buffer size results in a more efficient use of disk space and a corresponding less effective utilization of core storage.

Determination of Disk Buffer Size

The following guide rules may be used for determining the size of the disk buffer:

Rule 1. For random message requests, if the user plans to print out less than 10,000 characters per hour on a single 1053 Printer, the device utilization will be less than 20%. A large percentage of applications falls into this category. In this situation, for a single 1053 Printer, the user will almost never require more than three disk message buffer spaces.

Rule 2. If the user plans to print out a large number of messages in a small interval of time (e.g., data logging at 50 messages), he will require a large number of disk message spaces. The length of the log determines how big the disk buffer shall be.

The following example illustrates a representative calculation. Assume:

1. A 10 message-unit log at the end of every 15 minutes.
2. An average of 50 operational information message units per hour.
3. An average of 10 alarm message units per hour.
4. Message unit size for 1053 Printer (i.e., BZ1) = 50 words.
5. Average length of messages = 30 words.

To handle a 10 message-unit log will probably require 9 sectors; that is, assuming that all 10 messages are called to be typed at the same instant of time. The reason for the 9 is because 1 message is moved directly to the output area, the remaining 9 being buffered on disk.

Let us further assume that the remaining 60 message units are randomly distributed across the hour (that is, 10 in one 10 minute period, and perhaps none in the next 10 minute period, etc.).

Then,

$$\text{Number of characters typed} = 60 \times 30 \times 2$$

$$\text{Time to type these characters} = \frac{60 \times 30 \times 2}{15 \times 60} = 4 \text{ minutes}$$

The utilization of the 1053 Printer during the hour is

$$\frac{4}{60} = 6.67\%$$

Therefore, the number of sectors required for the messages sent at random is 3 (From Rule 1).

And the number of sectors required for the log is 9 (From Rule 2). Total number of sectors required is 9 + 3 = 12 sectors.

Now, assuming more than one typewriter is used, sum the number of sectors needed for each additional 1053 printer (computed as above). Let the total overall number of sectors = X.

Then $\text{NOCYL} = \dfrac{X + 7}{8}$ complete cylinders (ignore remainder)

In this example, $\text{NOCYL} = \dfrac{12 + 7}{8} = 2$ complete cylinders. If three extra 1053 printers were included to handle random message requests, then from Rule 2, six additional sectors will be required.

NOCYL now becomes $\dfrac{12 + 6 + 7}{8} = 3$ complete cylinders.

The user may also use the above guide rules to compute nonprocess disk buffering. Assuming a random message distribution pattern, each 1053 printer will require three sectors. Unless he has excessive disk storage, nonprocess disk requirements should be kept to a bare minimum.

CALCULATING TASK CORE SIZE

1. For an off-line system, the size of TASK is calculated as follows:

TASK = FIXED AREA
+ Disk device tables
+ DISKN
+ 1053 device tables
+ 1816 device tables
+ TYPEN
+ 1053-1443 Timing Response Routine
+ 1443 device table
+ PRNTN
+ Constants, work areas, etc.
+ CARDN (always included)
+ TASK Program Set

Where

TASK Program Set = 1690 + 8 X N + 653 X ONLIN
+ 200 X TRORG
+ 221 X TA01 X TRORG
+ 358 X TA02 X TRORG

+ 162 X TA03 X TRORG
+ 20 X MKLEV
+ 110 X DORG1

TRORG, ONLIN, TA01-3, DORG1, and MKLEV are TASK equate cards.

The remaining parameters have already been given in the calculation for Skeleton I/O: see System Design Considerations: System Director.

Once the system is built, the starting address of variable core is found at word 66 hexadecimal (102 decimal) of the Fixed Area. The label of this location is $VCOR. For an on-line system, the start address of variable core is equal to VCORE.

THE IBM NONPROCESS SYSTEM

The IBM Nonprocess System is a nonprocess system deck which constitutes the major portion of the TSX system. It is composed of control programs and a complete package of IBM relocatable subroutines necessary for the proper execution of the TSX system. A breakdown and brief description of each of its component parts in the order in which it is supplied and loaded to disk follows below (see Figure 62).

Cold Start Cards. The on-line or off-line system is brought into operation by three cold start cards (two Cold Start and one Name Card) which initiate the Cold Start program. A cold start requires that a

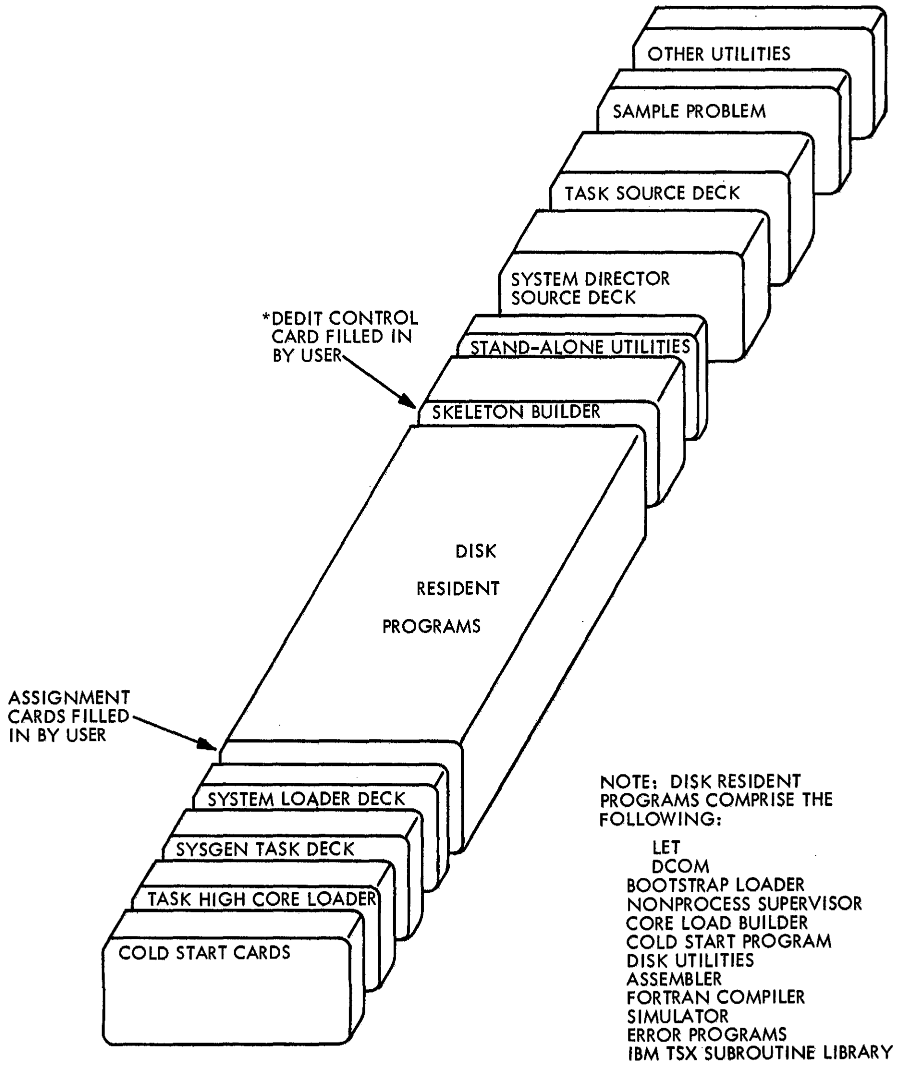


Figure 62. The IBM Nonprocess System

minimum of one core load be resident in the core load area on disk for execution. The name of the initial core load as well as the logical assignments of the physical disk drives are obtained from the Name Card.

System Generation (SYSGEN) TASK and Loader Cards. SYSGEN TASK is the "starter" system which contains the basic minimum components for initial system generation. It is loaded to memory by a four-card TASK High Core Loader.

System Loader. The System Loader performs three essential functions at system generation time: It 1) loads the IBM Nonprocess System to disk drive zero and file-protects this disk drive from sector 0 to the start of Nonprocess Work Storage, 2) builds the Assignment (AT) and Input-Output Unit (IOUT) Tables and stores them on disk and 3) edits the disk and the Disk Communications Area with a standard layout as a base for TSX non-process programs. It is also used for reload purposes and to make partial modification, if any, to the TSX system.

Disk LET/FLET Tables. LET (Location Equivalence Table) serves as a disk map for system programs, subroutines, and relocatable programs. It contains the name of each function and its size (that is, disk block count, where 1 disk block = 20 words). Each entry point in a subroutine has an entry in the LET table. As the user stores his own relocatable programs on the disk, entries for these programs are also made in LET.

FLET (Fixed Location Equivalence Table) is a map of core loads and data stored in the Process Core Image Storage (or Core Load) Area, and the Save Areas on disk.

Disk Communications Area (DCOM). DCOM is used by all nonprocess system programs and is stored on logical disk drive zero at sector 00000. It is essentially a disk communications map of vital information needed by nonprocess system programs. Some words within DCOM are used by process programs such as Cold Start.

This area is brought into core each time a // JOB is read; certain words are then initialized to reflect the current status of the disk as depicted by the LET/FLET tables. Whenever a // END or // XEQ card is encountered, DCOM is written back to disk.

Bootstrap for Nonprocess Supervisor. This is a relocatable program that can be located anywhere in core for any one system. When VCORE (the start address of variable core) is established, its entry point in variable core is fixed. The bootstrap serves as a linkage between the System Director or TASK, and the Nonprocess Supervisor. It is updated during system generation by TASK, the System Loader, and the Skeleton Builder program. It always resides on sectors 1 and 2 of logical disk drive zero.

Nonprocess Supervisor (SUP). This program directs all nonprocess monitor operations. It decodes the monitor control records in the stacked input for nonprocess jobs, and calls the appropriate monitor program (Assembler, FORTRAN Compiler, Simulator, etc.) to perform the desired operation. The supervisor provides continuous processor-controller operation during a sequence of jobs that might otherwise involve several independent programming systems. It also supervises the transfer of control between monitor and user programs.

Core Load Builder (CLB). This program constructs mainline nonprocess and interrupt core loads from user-written programs. Using data contained in control records and in the program itself, the Core Load Builder combines the mainline program, required subroutines, generated work area tables and transfer vectors into an executable core load.

Cold Start Program (CLST). This program initiates the TSX system into operation. In an on-line system, it loads the System Skeleton to core and transfers control to the System Director. In an off-line system, TASK is loaded to core, and control transferred to the first executable instruction within TASK.

Disk Utility Program (DUP). DUP is a set of routines designed to aid the user in performing the functions of disk maintenance. That is, it has the capabilities of storing, deleting, and outputting user programs, defining system and machine parameters, and also of maintaining communications areas. DUP also automatically updates the LET/FLET tables to reflect all changes to the disk. It is called into memory by the Nonprocess Supervisor.

Assembler (ASM). The Assembler is a disk-oriented symbolic assembly program that translates programs written in symbolic language into machine language. Basically, it is a one-for-one type assembly program. Provision is also included for the user to easily make use of input-output, conversion, and arithmetic subroutines that form a part of the subroutine library.

FORTRAN Compiler (FOR). This is a disk-oriented program that translates programs written in the FORTRAN language into machine language, and automatically provides for the calling of the appropriate arithmetic, functional, conversion, and input-output subroutines.

Simulator (SIM). The Simulator provides the user with the means for testing and debugging programs without disruption to the on-line process.

Error Programs. This is a collection of error subroutines called by the TSX Error Alert Control (EAC) program. EAC is executed when an internal or TSX detected error occurs.

Subroutine Library. The Subroutine Library is a package of IBM TSX and user-written subroutines resident in the relocatable subroutine area of disk. IBM TSX subroutines include: Real-time subroutines, Arithmetic and Functional subroutines, Conversion subroutines, FORTRAN I/O subroutines, and DP I/O subroutines.

Skeleton Builder. The Skeleton Builder uses tables constructed by the System Loader, user-assigned control records, and user-specified programs and subroutines to build the System Skeleton. The System Skeleton constitutes that portion of the system that remains in core during the execution of a TSX on-line system.

Stand-alone Utilities. These are optional utility routines which can only be loaded and executed under the control of TASK (in an off-line TSX system). The five utilities are: TASK Card to Disk, TASK Disk to Card, TASK Disk Patch, TASK Disk Duplication, and TASK Disk Loader.

System Director. This is the nucleus of the System Skeleton. It maintains control over the on-line process application by servicing all interrupts, handling error conditions, providing timer control over the process, and process program sequencing. The System Director is supplied as a source deck.

TASK. TASK is a "builder" operating system which controls the system generation process, and provides for the definition of the TSX system according to user specifications. It is supplied in source format.

Note that control programs are supplied assembled in absolute format; subroutines, in relocatable format. The System Director and TASK are the only exceptions: they are supplied as source decks.

In its original form, the IBM Nonprocess System does not contain those parameters which define and differentiate a system currently under construction from another, and is therefore unsatisfactory for direct use by a customer installation. Variability of interconnection of input-output devices is, however, permitted at the hardware level, and it is these variations which need to be communicated to the I/O subroutines if correct and intended operation is to be realized.

This communication is accomplished through the medium of the System Loader which accepts as input a statement of the system configuration (including the correlation between external device and interrupt identification, and internal hardware-sensed codes) and the IBM Nonprocess System master deck. To ready the IBM Nonprocess System for system loading, data from assignment cards is integrated into the master deck.

SYSTEM LOADER OPERATION

The System Loader assumes at system load time that only one disk drive (logical disk drive 0) is present on the system. After the IBM Nonprocess System is loaded, the user has the option of relocating certain disk areas (such as the Core Load Area, Process Work Storage, etc.) to an auxiliary disk drive or drives. This and other aspects of disk organization are discussed in System Design Considerations: Disk System Configuration.

Three essential functions are accomplished during a system load operation. These are:

- Loading the IBM Nonprocess System, including the subroutine library, to disk
- Building various TSX operating tables
- Editing the disk layout

Loading the IBM Nonprocess System

A typical sequence in which the input programs are loaded by the System Loader is given in Figure 63.

cc	1		3		5		7		9		11		13		15		17
	/	/	S	Y	S	T	E	M	L	O	A	D	E	R			
	*	A	S	S	I	G	N	M	E	N	T						
								.									
								.									
Assignment Cards								.									
								.									
								.									
	*	C	C	E	N	D		A	S	S	I	G	N	M	E	N	T
	*	L	D	D	S	K		.	L	E	T						
	*	L	D	D	S	K		.	D	C	O	M					
	*	L	D	D	S	K		.	S	U	P						
	*	L	D	D	S	K		.	C	L	B						
	*	L	D	D	S	K		/	C	L	S	T					
	*	L	D	D	S	K		.	D	U	P						
	*	L	D	D	S	K		.	A	S	M						
	*	L	D	D	S	K		.	F	O	R						
	*	L	D	D	S	K		.	S	I	M						
	*	L	D	D	S	K		.	E	P	R	G					
	*	C	C	E	N	D		S	Y	S	T	E	M				
	*	L	D	D	S	K		S	B	R	T						
	/	/	*		S	Y	D	I	R								
	*	C	C	E	N	D		S	B	R	T						
	*	D	E	D	I	T				K					C	Y		L

Figure 63. Sequence of Control Cards at System Load Time

Each program in the IBM Nonprocess System is preceded by an *LDDSK Control card which is read and analyzed by the System Loader. As a single sector at a time of a program is accepted, the appropriate sector address to which it is written on disk is determined by the first two words following a sector break record. A sector break record is a header record which serves two purposes:

- Enables the System Loader to establish a new disk sector either at a relative or absolute sector address

- Indicates if the phase of a program being read in involves either a principal I/O device or a principal print device, and, if any, which one.

Each phase within a program contains one sector break record. For example, since the FORTRAN Compiler is made up of 27 phases, it has 27 sector break cards. Sector break records are supplied in binary format (see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754. A separate discussion of sector break records is given at the conclusion of this section).

As each program is loaded to disk, an entry in the respective LET/FLET tables is updated accordingly. Note that the relocatable subroutine library may include user-written subroutines provided they are assembled/compiled by the TSX Assembler/FORTRAN Compiler. During the System Load stage, an error program within the System Loader ensures proper handling of error situations. Figure 64 reflects the layout of the IBM Nonprocess System on logical disk drive 0 after a system load operation.

Building the TSX Operating Tables

After the assignment cards have been read, two tables are built: 1) the Assignment Table, 2) the I/O Unit Table.

The input to the table-building phase are the assignment cards which are prepared by the user and merged with the IBM Nonprocess System.

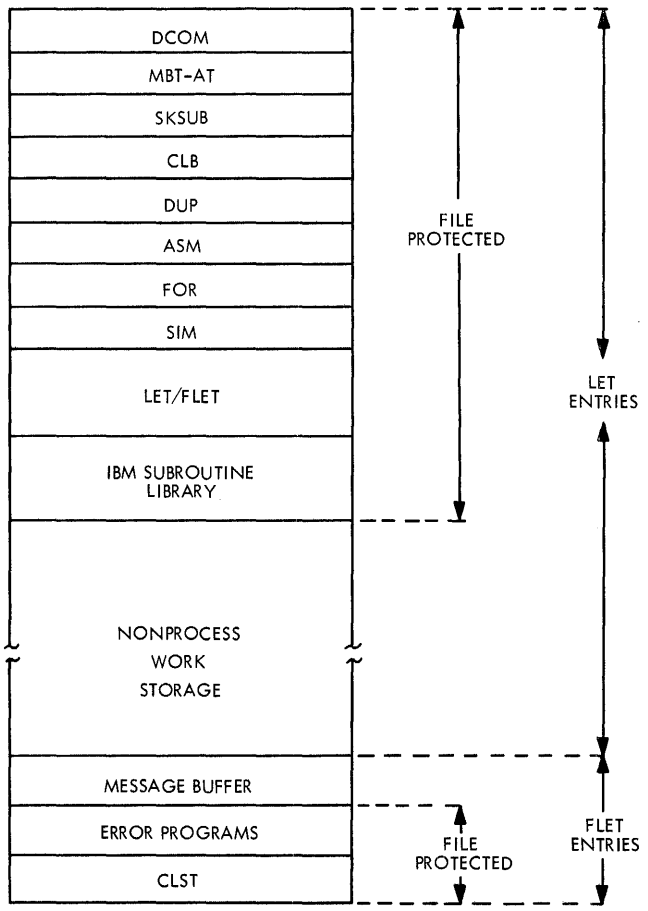


Figure 64. Disk Drive 0 after a System Load Operation

The Assignment Table (AT) serves to inform the Skeleton Builder (at Skeleton build time) which I/O device or PISW is assigned to a specific ILSW bit on a specific interrupt level. A 16-bit (IAC) code entry is furnished for each ILSW bit, which the Skeleton Builder later replaces by a branch address to transform it into the Master Branch Table (MBT). The (AT) table is stored on disk in reverse sequence; that is, level zero in highest location, etc. The number of AT entries and I/O interrupts are counted during the table build process and stored in sector 1 of logical drive zero.

The I/O Unit Table is constructed from the logical unit number (LUN) and/or its associated interrupt assignment code (IAC). The table is 44 words in length and is built in descending sequence; a maximum of 19 entries is allowed. The IOU Table is stored in the last 87 words of sector 2.

Editing the Disk Layout

The disk editing phase is entered after all absolute (or core image) and relocatable programs have been stored on disk, and the *DEDIT control card has been read.

The editing function initiates the disk and disk communications area with a standard layout as a base for TSX nonprocess programs. It uses LET/FLET and DCOM as communications areas.

In order to fix the boundaries of the various disk areas, certain information is required:

- Size of core of the Object Machine. This should be specified on the *DEDIT control card; otherwise, the source core size is construed as object core size.

- Size of Message Buffer. Note that the only area definition made by the user before the IBM Nonprocess System is loaded is the length of the message buffer. This must be specified on the *DEDIT control card and should correspond to NOCYL (TASK equate card) at TASK assembly time. The calculation of message buffer size is discussed at some length in the section System Design Considerations: TASK.

- Size of IBM Nonprocess System areas. These are made known to the System Loader after the system is loaded to disk.

Note that the boundaries of the following areas:

Nonprocess Save Area
Process Save Area
Special Save Area

depend on the estimated size of the System Skeleton (see System Design Considerations: Disk System Configuration).

LET/FLET Entries. Fixed entries, derived from control cards, exist in LET for the following:

Disk Communications Area (DCOM)
Master Branch Table/Assignment Table (MBT/AT)
Skeleton Subroutine Map (SK-SUB)
Nonprocess Supervisor (SUP)
Core Load Builder (CLB)
Disk Utility Program (DUP)
Assembler (ASM)
FORTRAN Compiler (FOR)
Simulator (SIM)

An entry for each subroutine is made while it is being loaded.

FLET entries, on the other hand, are made from computed and assumed sizes for the following:

Cold Start
Error Programs
Message Buffer

After these FLET entries have been made, the .E entry of LET is updated to reflect the boundaries of the Nonprocess Work Storage for the remaining disk space available.

DCOM Entries. The first sector address of each of the following areas are entered in DCOM:

Nonprocess Supervisor (SUP)
Disk Utility Program (DUP)
Assembler (ASM)
FORTRAN Compiler (FOR)
Simulator (SIM)
Location and Fixed Location Equivalence Tables (LET/FLET)
Nonprocess Work Storage (NPWS)

FUNCTION OF THE *ASSIGNMENT CARDS

The assignment card serves to assign the various I/O devices and machine functions to a particular interrupt level and bit. Assignments are in the form of interrupt assignment codes (IAC) which are fixed for each device, and logical unit numbers (LUN) which are selected by the user for linkage to user-written FORTRAN programs.

Through the assignment card, the user

1. Assigns IAC codes to the various interrupt levels and ILSW bits (within the level used on the system).
2. Assigns LUN numbers as they are used in user-written FORTRAN programs, to certain data processing input-output (DP I/O) devices by equating them to corresponding IAC codes.

Interrupt assignment codes uniquely define all process interrupts, I/O devices, console interrupts, and interval timers. They are fixed and may not be changed by the user. Their values range from 00 through 63.

Logical unit numbers on the other hand are used to identify DP I/O devices in user-written FORTRAN programs, and are specified by the user at system load time. The LUN's are entered into the I/O Unit Table to permit communication of FORTRAN programs with FORTRAN I/O at object time. Once fixed, they cannot be changed without repeating the Assignment Table building phase of the System Loader and Skeleton rebuild under certain conditions, as well as the recompilation of every user-written FORTRAN program utilizing DP I/O devices affected.

A maximum of 19 different LUN's is possible on a TSX system with a full complement of I/O devices. LUN values range from 01 through 44. Note that no LUN may be assigned to more than one particular device. In a minimum (8K) TSX system, it is advisable, for purposes of space conservation, to use the lowest LUN numbers first, since the System Loader will build a table providing space for all LUN's up to the largest number assigned. Keeping LUN numbers small, therefore, conserves core storage. The reader should refer to IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754 for details of assignment card formats and operational procedures.

Examples of the Use of LUN Numbers/IAC Codes.

Consider the following assignment cards:

EXAMPLE 1.

```
Ø3  Ø6  Ø1/41,Ø2,37/Ø1,Ø5,33,Ø8
```

Level 3 contains 6 ILSW bits. IAC 01 represents an 1816/1053 printer which has a LUN of 41 assigned to it, while IAC 02, representing a 1442 card/read punch, has the same LUN as its IAC; that is, it requires no LUN entry. The combination 37/01 represents another printer to which a LUN of 01 is assigned by the user; IAC 05 represents a 1627 plotter unit with the same LUN number as its IAC code (that is, 05). IAC 33 represents a process interrupt. IAC 08 represents a 2310 disk drive which has no assignable LUN number.

EXAMPLE 2.

```
99  Ø2  43/Ø1,44/Ø9
```

The 1816 keyboard on group 2 has a LUN of 1 while the second magnetic tape drive has a LUN of 9.

EXAMPLE 3.

```
//SYSTEMLOADER
*ASSIGNMENT
00 02 33,00
01 04 33,04,08,09
02 02 33,14
03 05 33,01,36,37,38
04 03 33,34,35
05 05 33,10,16,11,12
06 02 33,06/03
07 02 33,02
08 03 33,32,05/07
09 01 33
10 01 33
11 01 33
*CCEND ASSIGNMENT

DEVICE    LEV    BIT    IAC    LUN

PISW      00     00     33
TIMERS    00     01     00
PISW      01     00     33
DISK-1    01     01     04
DISK-2    01     02     08
DISK-3    01     03     09
PISW      02     00     33
MAGT-1    02     01     14     14
PISW      03     00     33
TYP1G1    03     01     01     01
TYP2G1    03     02     36     36
TYP3G1    03     03     37     37
TYP4G1    03     04     38     38
PISW      04     00     33
COMP-1    04     01     34
COMP-2    04     02     35
PISW      05     00     33
ADC-1     05     01     10
ADC-2     05     02     16
DINP      05     03     11
DAOP      05     04     12
PISW      06     00     33
PRNT-1    06     01     06     03
PISW      07     00     33
CARD-1    07     01     02     02
PISW      08     00     33
CONSOL    08     01     32
PLOT-1    08     02     05     07
PISW      09     00     33
PISW      10     00     33
PISW      11     00     33

 YOU DEFINED 000018 I/O DEVICES
 AND A TOTAL OF 000031 ILSW BITS
```

This illustrates an example of user assignment of I/O devices and process interrupts to 12 levels of interrupts defined in a sample machine configuration given in System Design Considerations: System Director.

Note that only two I/O devices have been assigned LUN numbers:

1627 Plotter (IAC = 05)	=	07
1443 Printer (IAC = 06)	=	03

The remaining devices use their IAC codes (a user option) as LUN's. Note also that process interrupts and certain DP I/O devices have no assignable LUN's. The map correlates each process interrupt or device with its level, bit, IAC code, and LUN (if any).

Note that IAC/LUN groups may contain either the IAC code alone or a combination of the IAC code and the LUN as assigned by the user to that IAC (and separated by a slash). When the LUN number is omitted, it means that either no LUN is defined (that is, not assignable) or that the System Loader considers the LUN to be identical to the IAC code. The user has the option of assigning the value of the corresponding IAC code to the LUN for a particular device.

Devices with no Interrupt-entry on any Level

The 1816 Keyboard units on printer groups 1 and 2 and the second Magnetic Tape drive have no separate defined interrupts, their interrupts being the same as that of the first 1816/1053 printer and first magnetic tape drive respectively. However, a LUN has to be assigned to them whenever they are used in connection with FORTRAN programs. In these three special cases, a dummy interrupt level number 99 is defined, followed by a standard format entry for bit count and IAC code. The dummy level 99 can be omitted should all three possible devices have a LUN identical to their IAC code.

THE *DEDIT CONTROL CARD

The *DEDIT Control card starts the disk editing phase: that is, it starts the function of editing the layout of the disk during which time the System Loader uses LET/FLET and DCOM as communications areas. Some of the activities carried out during this phase include (see Editing the Disk Layout):

1. Initialization of the FLET area on disk
2. Calculation of the source core size
3. Entry of the object core size into the disk communications area (DCOM)
4. Entry of message buffer size in cylinders into DCOM
5. File protection of the IBM Nonprocess System

Parameters

Two important parameters must be specified by the user:

1. Size of core of the object machine
2. Size of message buffer size

The calculation of the core size of the source machine (that is, the machine on which the IBM Nonprocess System is loaded) is achieved by TASK and the result is stored in the Fixed Area in core. The System Loader then places this result in DCOM. The user may exercise the option to define a different core size for the object machine (that is, the machine on which the TSX system is executed). This will also be stored in DCOM. If the object core size is not specified on the *DEDIT card, the source core size will serve as object core size.

As noted earlier, the only area definition made by the user before the IBM Nonprocess System is loaded is the definition of the length of the disk message buffer. This is specified in cylinders in the *DEDIT card and must equal NOCYL (TASK equate card). The calculation of the size of the message buffer is discussed in detail in System Design Considerations: TASK.

An example of the use of the *DEDIT card is given below:

```
*DEDIT 16K 011CYL

THE SOURCE CORE-SIZE IS    016384
THE OBJECT CORE-SIZE IS    016384
```

The *DEDIT control card is the last card recognized by the System Loader.

Reentering the Disk Edit Phase

The disk editing function permits a reentry by the user after the IBM Nonprocess System is loaded and control returned to SYSTEM TASK. This may be needed for:

1. Rebuilding the FLET table.
2. Changing the Message Buffer Size.
3. Changing Object Core Size.
4. Changing the assignment of LUN numbers, such as, for example, if an error was made in the user-assignment of an IAC or a LUN.

SUMMARY OF ASSIGNMENT CARD RESTRICTIONS

Assignment designation is governed by the following rules:

1. A separate assignment card is used for each interrupt level. Assignment cards may be in any order of interrupt level number.
2. The number of IAC/LUN codes specified per level must be equal to the number of interrupt level status word (ILSW) bits used.
3. Only the IAC code 33 (for process interrupts) may be used more than once. In the case of LUN numbers, the same LUN cannot be assigned to more than one device, nor can a device have more than one LUN assigned to it.
4. For IAC codes 42, 43, and 44, a dummy interrupt level entry of 99 must be specified. These refer to the 1816 keyboards on printer groups 1 and 2 and the second magnetic tape drive.
5. For RPQ devices, IAC codes 20 - 31 and 45 - 63 may be used. In any TSX system, IAC codes 00, 02, 04, and 32 must be used; 01 or 06 must also be used.
6. If more than one group of process interrupts are assigned to a particular level, the second group must be treated as an RPQ device, given an RPQ IAC code and a user-written ISS subroutine to accommodate this device. The subroutine will indicate to the System Loader which IAC code it responds to; it will have to be core-resident at all times.

SECTOR BREAK RECORDS FOR ABSOLUTE PROGRAMS

Absolute programs are generated by an absolute 1800 assembly and are loaded by the System loader, one record at a time (taking into account all data breaks and origin changes), to disk in true Core-Image Format. That is, each program resides on disk in exactly the same format in which it will reside in core storage. Core-Image Format is also called Data Format because a program thus stored on disk can be transferred to core by a single call (to DISKN) without any data manipulation. All IBM system programs (e.g., Assembler, FORTRAN Compiler, Simulator) are stored in this format.

However, it is from a header or sector break record that the absolute loader portion of the System Loader determines the sector address at which succeeding data is to be stored. The sector within which the data is to be stored is first read into core, one word of data at a time, until that sector in core is completed. When full, the 320-word sector buffer is written to disk and the next sector break record is read to locate the next sector to be written.

Four types of Sector Break Records are used by the System Loader:

Type 1
Type 2
Type 9
Type E

Note that Type F cards are "trailer" or "transfer" cards which occur at the end of a binary deck. The format of each card is in the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

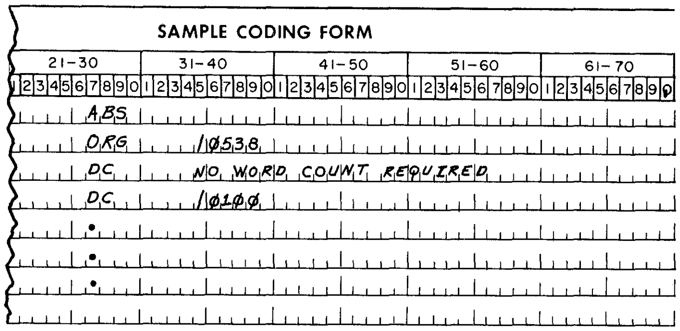
From each type of sector break record, the System Loader interprets the sector address as follows:

Type 1:	As an absolute address.
Type 2:	As a displacement from the last sector loaded.
Type 9:	As a displacement from the last absolutely defined sector address (that is, defined by a Type 1 sector break record

Type E is a special sector break record type used only by the Simulator subroutine package. It is treated by the System Loader like a Type 1, except that it causes data to be streamed to the disk contiguously, ignoring data breaks brought about by BSS's; or by an ORG to the same location as the data card immediately following the Type E record.

The Type 1 sector break record is generated by an ABS statement in an absolute assembly; Type 9 or Type E records must be inserted manually in the object deck by the user. Type 2 sector break records are generated during 1) assembly of mainline-type programs without an ABS statement, and 2) FORTRAN compilation of mainline programs.

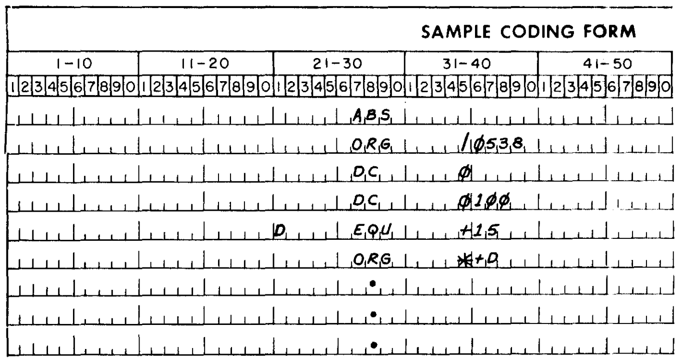
The sector address on disk to which the System Loader begins writing the program is defined in the second word of the program following the ABS and first ORG statements. The first word may contain any value; no word count is required. An example is given below:


SAMPLE CODING FORM

```
      ABS
      ORG     /Ø538
      DC      NO WORD COUNT REQUIRED
      DC      /Ø1ØØ
       •
       •
       •
```

The first DC is at location /0538, the second at /0539. The System Loader will start to load at sector /0100. The first word of the sector is at /053A -- the content of /0538 is not loaded to disk since it does not constitute an integral part of the program. If this program was later called from the disk, the word count and sector address would be specified by the AREA (portion) of the disk call required at /0538 for proper execution.

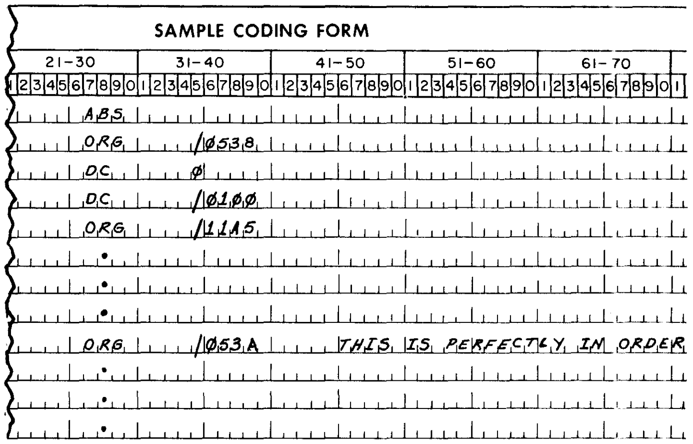
If the first two words of the program are followed by another ORG statement, as shown below, the program will be placed in a location on disk reflecting a displacement from the address defined by the first ORG.


SAMPLE CODING FORM

```
                          ABS
                          ORG     /Ø538
                          DC      Ø
                          DC      Ø1ØØ
                    D     EQU     +15
                          ORG     *+D
                           •
                           •
                           •
```

The program will be loaded starting at position 15 of sector /0100, leaving positions 0 through 14 at whatever value they previously had on disk. The displacement D could have any value -- thus skipping over several sectors.

Note, however, that D cannot have a negative value. This is a necessary requirement of the TSX system which is designed such that it is impossible to inadvertently destroy a program residing on preceding sectors by back-origining. Thus, the lowest origin in the program is required to be immediately after the ABS statement. As shown below, this constitutes, in no way, a system limitation. Note that normal back origins, as they occur in every program, are perfectly legal.


SAMPLE CODING FORM

```
      ABS
      ORG     /Ø538
      DC      Ø
      DC      /Ø1ØØ
      ORG     /11A5
       •
       •
       •
      ORG     /Ø53A          THIS IS PERFECTLY IN ORDER
       •
       •
       •
```

The program will be correctly located on disk reflecting exactly the layout in core.

The following example illustrates an ERROR STOP (System Loader error message - LO5) situation.

SAMPLE CODING FORM

```
      ABS
      ORG     /Ø538
      DC      Ø
      DC      /Ø1ØØ
       .
       .
       .
      ORG     /Ø4ØØ     THIS IS NOT ALLOWED
       .
       .
```

This is clearly illegal because whatever program that was residing on sector /00FF would be destroyed.

The final example below illustrates Type E sector break record functioning:

SAMPLE CODING FORM

```
                          ABS
                          ORG     /Ø538
                          DC      Ø
                          DC      /Ø1ØØ
                           .
                           .
                           .
                          ORG     /Ø538
                          DC      Ø
                          DC      /Ø1ØØ
                           .
                           .
                           .
```

If this sequence is preceded by a Type 1 or Type 9 sector break record, the data up to the second ORG would not appear on disk, but would be overlaid by the second sequence. If, however, the sequence is preceded by a Type E card, all the data would appear on disk, and the data following the second ORG statement would immediately follow the first with no sector break being forced.

SYSTEM DIRECTOR

When the IBM Nonprocess System is loaded, assignment cards supply the System Loader with data which relate to the interrupt level allocation of I/O units, process interrupts, interval timers, etc. That is, they provide a statement of the system and interrupt configurations.

At System Director assembly time, the System Director must also be tailored to meet the exact requirements set by the user. These requirements include:

1. Definition of the size of the System Director
2. Definition of functions required
3. The allocation of internal and external interrupt levels
4. The number of CALL Count routines to be included by the user

Since the System Director must be assembled and stored on disk before the TSX System Skeleton can be built, some means must be employed to make the System Director aware of these requirements. To achieve this, a set of System Director EQUATE cards (provided pre-punched by IBM) is prepared by the user and physically placed in the System Director source deck. The resulting integrated deck is then assembled under the control of an off-line nonprocess monitor. Figure 65 depicts the physical relationship of the EQUATE cards to the System Director Source Deck.

Figure 66 illustrates an example of a complete set of System Director EQUATE cards. In terms of definition requirements, the set can be broken down into convenient subsets as follows:

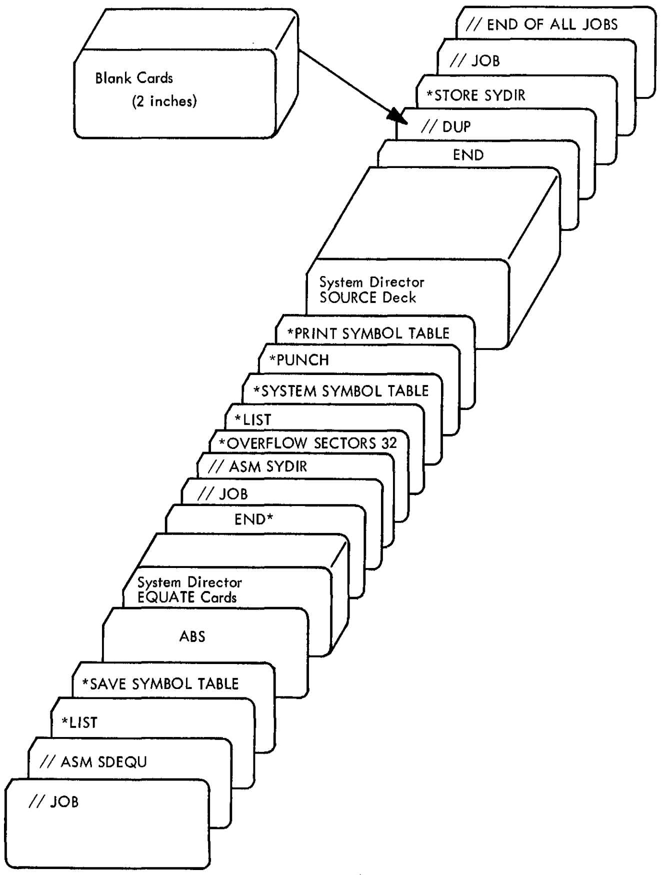


Figure 65. System Director Source Deck and EQUATE Cards

a) Size of System Director	VCORE, NUQUE; also a function of b), c), and d)
b) Functions required (Interval Timer Control, Time-Sharing, Operations Monitor, and Error Alert Control (EAC) DUMP)	ITCUS, TBASE, CBASE, TIME1-2, TISHA; TIMES; OPMOI and DUMP1
c) Allocation of internal and external interrupt levels	NULEV, USE00-23, NB00-23, NIL00-23, NLWS1-2, and ICLL1-2
d) Number of CALL Count routines required by user	NITP1-2

SIZE OF SYSTEM DIRECTOR

Since the System Director is a component part of the System Skeleton, it must be core-resident at all times in an on-line system in order to respond to the real-time world. Its required core size will, however, vary according to the user's machine configuration, process requirements, and other options.

For example, if the user specifies when the system is assembled that time-sharing is to be used, the Time-Sharing Control (TSC) program will be included in core. If he has no use for time-sharing, TSC may be eliminated.

Similarly, if the user specifies that interval timers are not used, the Interval Timer Control (ITC) program as well as TSC may be eliminated. It is a rule, however, that ITC must be in core if time-sharing is utilized. The Program Sequence Control (PSC), Error Alert Control (EAC), and Master Interrupt Control (MIC) programs must always be used, but each is variable in size according to the number of interrupt levels elected.

In addition, a work area is associated with each interrupt level; for example, if the user elects 12 levels of interrupts, 12 work areas are required; if he elects 24 levels, the System Director will require 24 work areas. Three other additional work areas are included: one each for Error, Mainline, and Nonprocess.

```
NULEV EQU    12          NUMBER OF LEVELS USED      SYD00060
USE00 EQU    1           1-LEVEL USED   0-NOT USED  SYD00070
USE01 EQU    1           1-LEVEL USED   0-NOT USED  SYD00080
USE02 EQU    1           1-LEVEL USED   0-NOT USED  SYD00090
USE03 EQU    1           1-LEVEL USED   0-NOT USED  SYD00100
USE04 EQU    1           1-LEVEL USED   0-NOT USED  SYD00110
USE05 EQU    1           1-LEVEL USED   0-NOT USED  SYD00120
USE06 EQU    1           1-LEVEL USED   0-NOT USED  SYD00130
USE07 EQU    1           1-LEVEL USED   0-NOT USED  SYD00140
USE08 EQU    1           1-LEVEL USED   0-NOT USED  SYD00150
USE09 EQU    1           1-LEVEL USED   0-NOT USED  SYD00160
USE10 EQU    1           1-LEVEL USED   0-NOT USED  SYD00170
USE11 EQU    1           1-LEVEL USED   0-NOT USED  SYD00180
USE12 EQU    0           1-LEVEL USED   0-NOT USED  SYD00190
USE13 EQU    0           1-LEVEL USED   0-NOT USED  SYD00200
USE14 EQU    0           1-LEVEL USED   0-NOT USED  SYD00210
USE15 EQU    0           1-LEVEL USED   0-NOT USED  SYD00220
USE16 EQU    0           1-LEVEL USED   0-NOT USED  SYD00230
USE17 EQU    0           1-LEVEL USED   0-NOT USED  SYD00240
USE18 EQU    0           1-LEVEL USED   0-NOT USED  SYD00250
USE19 EQU    0           1-LEVEL USED   0-NOT USED  SYD00260
USE20 EQU    0           1-LEVEL USED   0-NOT USED  SYD00270
USE21 EQU    0           1-LEVEL USED   0-NOT USED  SYD00280
USE22 EQU    0           1-LEVEL USED   0-NOT USED  SYD00290
USE23 EQU    0           1-LEVEL USED   0-NOT USED  SYD00300
NB00  EQU    2           1+HIGHEST ILSW BIT = USED  SYD00310
NB01  EQU    4           1+HIGHEST ILSW BIT = USED  SYD00320
NB02  EQU    2           1+HIGHEST ILSW BIT = USED  SYD00330
NB03  EQU    3           1+HIGHEST ILSW BIT = USED  SYD00340
NB04  EQU    5           1+HIGHEST ILSW BIT = USED  SYD00350
NB05  EQU    5           1+HIGHEST ILSW BIT = USED  SYD00360
NB06  EQU    2           1+HIGHEST ILSW BIT = USED  SYD00370
NB07  EQU    2           1+HIGHEST ILSW BIT = USED  SYD00380
NB08  EQU    3           1+HIGHEST ILSW BIT = USED  SYD00390
NB09  EQU    1           1+HIGHEST ILSW BIT = USED  SYD00400
NB10  EQU    1           1+HIGHEST ILSW BIT = USED  SYD00410
NB11  EQU    1           1+HIGHEST ILSW BIT = USED  SYD00420
NB12  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00430
NB13  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00440
NB14  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00450
NB15  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00460
NB16  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00470
NB17  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00480
NB18  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00490
NB19  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00500
NB20  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00510
NB21  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00520
NB22  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00530
NB23  EQU    0           1+HIGHEST ILSW BIT = USED  SYD00540
NIL00 EQU    1           1+HIGHEST PISW BIT = USED  SYD00550
NIL01 EQU    5           1+HIGHEST PISW BIT = USED  SYD00560
NIL02 EQU    9           1+HIGHEST PISW BIT = USED  SYD00570
NIL03 EQU    13          1+HIGHEST PISW BIT = USED  SYD00580
NIL04 EQU    1           1+HIGHEST PISW BIT = USED  SYD00590
NIL05 EQU    5           1+HIGHEST PISW BIT = USED  SYD00600
NIL06 EQU    9           1+HIGHEST PISW BIT = USED  SYD00610
NIL07 EQU    13          1+HIGHEST PISW BIT = USED  SYD00620
NIL08 EQU    1           1+HIGHEST PISW BIT = USED  SYD00630
NIL09 EQU    16          1+HIGHEST PISW BIT = USED  SYD00640
NIL10 EQU    16          1+HIGHEST PISW BIT = USED  SYD00650
NIL11 EQU    16          1+HIGHEST PISW BIT = USED  SYD00660
NIL12 EQU    0           1+HIGHEST PISW BIT = USED  SYD00670
NIL13 EQU    0           1+HIGHEST PISW BIT = USED  SYD00680
NIL14 EQU    0           1+HIGHEST PISW BIT = USED  SYD00690
NIL15 EQU    0           1+HIGHEST PISW BIT = USED  SYD00700
NIL16 EQU    0           1+HIGHEST PISW BIT = USED  SYD00710
NIL17 EQU    0           1+HIGHEST PISW BIT = USED  SYD00720
NIL18 EQU    0           1+HIGHEST PISW BIT = USED  SYD00730
NIL19 EQU    0           1+HIGHEST PISW BIT = USED  SYD00740
NIL20 EQU    0           1+HIGHEST PISW BIT = USED  SYD00750
NIL21 EQU    0           1+HIGHEST PISW BIT = USED  SYD00760
NIL22 EQU    0           1+HIGHEST PISW BIT = USED  SYD00770
NIL23 EQU    0           1+HIGHEST PISW BIT = USED  SYD00780
NLWS1 EQU    12          NO. PROG. INT. GROUP 0-13  SYD00790
NLWS2 EQU    0           NO. PROG. INT. GROUP 14-23 SYD00800
NITP1 EQU    16          NO. COUNT SUBRS. GROUP 1   SYD00810
NITP2 EQU    8           NO. COUNT SUBRS. GROUP 2   SYD00820
ICLL1 EQU    /007F       INT. CORELOAD LEVEL MASK   SYD00830
ICLL2 EQU    /FFFF       INT. CORELOAD LEVEL MASK   SYD00840
ITCUS EQU    1           1-ITC USED   0-NOT USED    SYD00850
TBASE EQU    -500        CLOCK BASE=MILSEC*TBASE    SYD00860
CBASE EQU    1           COUNT BASE=MILS*TBASE*CBASESYD00870
TIME1 EQU    /0000       TIMER C MILS*TBASE         SYD00880
TIME2 EQU    /07D0       TIMER C MILS*TBASE         SYD00890
VCORE EQU    10000       ADDR. 1ST WORD VARIABLE CORSYD00900
NUQUE EQU    50          NUMBER OF QUEUE ENTRIES    SYD00910
DUMP1 EQU    1           1-EAC DUMP USED 0-NOT USED SYD00920
OPMO1 EQU    1           1-ITC RESETS 0-USER RESETS SYD00930
TISHA EQU    32767       TIME-SHARING PERIOD        SYD00940
TIMES EQU    1           1-TSC USED 0-TSC NOT USED  SYD00950
```

Figure 66. Example of a Set of System Director Equate Cards

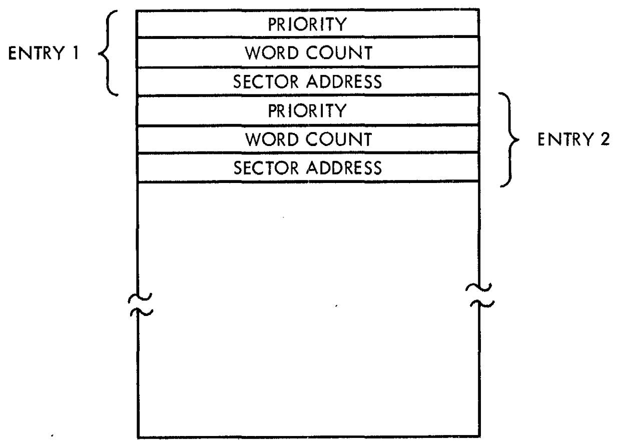


Figure 67. Mainline Core Load Queue Table

Mainline Core Load Queue Table

Resident within MIC is a Queue Table made up of three-word entries used for the stacking of mainline core loads requested for execution, as shown in Figure 67.

Each time the QUEUE routine is called, an entry is made in the queue if there is not a like entry of equal priority and sector address already in the queue. Entries are removed from the Queue Table by the subroutines UNQ and VIAQ (see Program Scheduling).

The size of this table -- that is, its maximum number of entries -- is specified by the user on the NUQUE equate card. It should be large enough so that the Queue Table shall not overflow under normal operating conditions.

VCORE determines the starting address, which must always be even, of the variable core area. The appropriate value of VCORE can be arrived at by calculating the size of the System Director, Skeleton I/O and the user-written subroutines.

Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

	1116	(constant for MIC, PSC, and EAC and their work areas)
+	220	(if ITC is included: that is, when ITCUS = 1)
+	95	(if EAC dump is required: that is, when DUMP1 = 1)
+	109	multiplied by the number of interrupt levels (that is 109 x NULEV)
+	3	multiplied by the number of Queue entries (that is, 3 x NUQUE)
+	2	multiplied by the number of process interrupts (that is, 2 x sum of NIL00 through NIL23)
+	2	multiplied by the number of programmed interrupts on levels 0 through 13 (that is, 2 x NILSW1)
+	2	multiplied by the number of programmed interrupts on levels 14 through 23 (that is, 2 x NILSW2)
+	2	multiplied by the number of count subroutines 0-15 (that is, 2 x NIPT1)
+	2	multiplied by the number of count subroutines 16-31 (that is 2 x NIPT2)
+	334	(if TSC is included: that is, when TIMES = 1)
+	66	(if more than 14 levels are used)
+	6	(if more than 14 levels are used and ITC is included)
+	8	(if more than 14 levels and TSC is included)

From the configuration set out in Figures 66 and 68, a typical calculation is deduced below.

System Director Core Size	=	1116
	+	220
	+	95
(109 x 12)	+	1308
(50 x 3)	+	150
(2 x 57)	+	114
(2 x 12)	+	24
(2 x 0)	+	0
(2 x 16)	+	32
(2 x 8)	+	16
TSC	+	334
		3409 words

DEFINITION OF FUNCTIONS REQUIRED

Interval Timer Control

When the ITCUS label is equated to 1, the ITC program is included within the System Director and serves to set up user-specified times and correct linkages to the user's subprograms. Once this is done, ITC will control the timers until one or more specified intervals have elapsed, at which point control is transferred to a user's subprogram.

Specifications for any timer may be set or changed in relation to the timer base at any time during an on-line process operation by the calling sequence.

It was mentioned in Functions of Executive Programs: The System Director, that a programmed real-time clock, a time-sharing control timer, and nine programmed interval timers are controlled (that is, updated) by the third machine interval timer C. It is, however, the user's responsibility at assembly time to establish:

1. A primary time base (TBASE) for the real-time clock; that is, how often the clock should be updated.
2. A secondary time base (CBASE) for the programmed timers and time-sharing control timer.

Primary Time Base

This is that interval of time used to update the real-time clock, and is called the Interrupt Time Base. It is the product of the wired-in hardware time base and a number chosen by the user (TBASE) at assembly time, expressed as follows:

INTERRUPT TIME BASE = (WIRED-IN HARDWARE TIME BASE) X (USED-ASSIGNED NUMBER)

For example, if the machine interval timer C is wired for a four millisecond time base and the real-time clock is to be updated every two seconds, the user-assigned number can be calculated to be 500. TBASE is thus equated to minus (-) 500. A negative number is used because the interval timer is incremented in the positive direction, causing an interrupt when zero is reached. The primary time base for the real-time clock in this example (that is, how often it is to be updated) is thus two seconds.

To enable ITC to keep track of elapsed time since the last or previous interrupt occurred, a double-word TIME1 and TIME2 is equated to the hexadecimal equivalent of the interrupt time base. This value is added to the real-time clock each time an interval timer C interrupt occurs.

In the above example, TIME1 and TIME2 are equated to /0000 and /07D0. The label TIME1 is always /0000 unless the calculated interrupt time base exceeds 65,535 milliseconds.

Secondary Time Base

The programmed timer base for the nine programmed timers and time-sharing control timer is a user-assigned multiple of the interrupt time base established for the real-time clock, and expressed as follows:

PROGRAMMED TIMER BASE = (INTERRUPT TIME BASE) X (USER-ASSIGNED NUMBER)

For example, if the interrupt time base is fixed at two seconds, and the user wants the programmed timers to operate at 30-second intervals, the label CBASE is equated to 15.

This base is used specifically for the nine programmed timers and the time-sharing control timer, and is the smallest interval of time that can be specified for the programmed timers or for time-sharing operations.

Time-Sharing

The TIMES label specifies at assembly time whether or not time-sharing is to be used.

It was noted in the preceding section that the programmed timer base is the smallest interval of time that can be specified for programmed timing or time-sharing operations. When time-sharing is used, a user-assigned multiple of the programmed timer base is established.

For example, if the programmed timer base is fixed at 30 seconds and the user desires time-sharing operations of two minutes' duration whenever the queue is empty, the label TISHA is equated to 4. Thus, the time-shared operation is terminated whenever the time interval specified (in this case two minutes) has elapsed. TISHA is identical to the parameter I in the requesting CALL SHARE statement in the mainline program. If the user wishes to remain in time-sharing until some core load name is put into the queue by an interrupt program which uses CALL ENDTS, then TISHA may be specified for the longest possible numerical value, that is, 32767. The reason for this is to keep the time-sharing function from exchanging core unnecessarily at frequent intervals to check the Queue Table when no entries have been put in the queue. This is the recommended procedure.

Operations Monitor

The user may select an option in ITC to reset the Operations Monitor (a hardware feature) during nonprocess operations. He does this by equating the OPMOI card to 1 or 0: a 1 indicates that the monitor is to be reset by ITC; a 0 indicates that the monitor is to be reset by user program control. It should be noted that the Nonprocess Monitor does not incorporate the Operations Monitor reset instruction. ITC will only execute the reset if time-sharing is in progress.

Error Alert Control (EAC) Dump

The label DUMP1 gives the user the option of including the dump routine (dump core to disk) for subsequent user error analysis. The functions of EAC are explained in another section of this manual (see Functions of Executive Programs: The System Director).

ALLOCATION OF INTERNAL AND EXTERNAL INTERRUPT LEVELS

Interrupts can be generated by events which originate in the plant or the environment that is being controlled, or by conditions internal to the computer hardware itself. These may be classified as external (or process) interrupts and internal interrupts.

Internal interrupts may be caused by an error condition being detected, an input/output operation being completed, an interval timer interrupt, a computer operator setting a switch, etc.

External or process interrupts may be caused by the closing of an electrical contact, a rise in temperature above a set limit, etc.

Since the number of internal and external interrupts required by a particular system is decided by the user, the System Director must be provided with a labelled assignment of each interrupt used.

Interrupt Levels

A level of interrupt represents a degree of removal from the normal computer mode. The multi-interrupt feature of the IBM 1800 Data Acquisition and Control System is composed of a maximum of 24 levels, each level containing 16 request positions, thus making available 384 interrupt lines to signal the computer to halt the program being executed and branch to unique hardware memory locations.

The number of interrupt levels (NULEV) planned by a user is assigned contiguously to the 24 available levels, starting from zero to 23. If, for example, 16 interrupt levels are elected by the user, levels 0-15 are used. The numerical value to which the label NULEV is equated is always 1 plus the highest numbered interrupt level used.

Priority assignments are necessary in order that an order of precedence (that is, a level) can be established among the several interrupt conditions. In configurating a multi-interrupt system, the user should remember that certain I/O devices such as the disk, magnetic tape, and timers require high response capabilities. Other I/O devices such as the list printer, typewriter, and card-reader do not demand such a critical response.

In general, process interrupts (PISW's) are assigned lower priority levels than data processing and process I/O devices, except for process interrupts that do not require I/O and demand immediate response or initiate extended operations at lower levels through the programmed interrupt feature. The reason process interrupts are assigned lower priorities than I/O devices is that user-written subroutines for the servicing of these process interrupts can then utilize all I/O devices. I/O devices must receive an operations complete interrupt, which cannot occur if it is located on a lower priority level than the level from which the I/O device is called. Exceptions to this rule are the disk and the 1053 Printer where the I/O routine is so written that it will remain within itself until the operation is complete. These exceptions were allowed due to EAC requirements, but should not, in general, be considered as acceptable practice.

The amount of computer time required to service a particular interrupt can influence its priority assignment. If, for example, its servicing is relatively short, an interrupt may be accorded higher priority than one which entails more elaborate servicing procedures.

Those basic I/O devices that demand fast response include the disk, magnetic tape, and timers. Because the 1053 Printer uses the disk when it buffers messages, the analog interrupts should be at a higher level than the assignment of the 1053 Printers due to a possible loss of comparator interrupts. It should be pointed out that although fast response is not normally required by the 1053 Printer, this device should be assigned to a high enough interrupt level to allow it to run continuously at a maximum rate. Thus, typewriter messages will be serviced without overloading the message buffer.

It is recommended that the Analog Input Comparator feature be assigned to a higher priority level than the Analog Input. The remaining I/O devices do not possess any special characteristics for assignment at a high level, except that they must be at a level higher than the highest level from which they are called, and at a higher level than any assigned interrupt core load (see equate card ICLL1, Figure 66).

Figure 68 (in conjunction with Figure 66) illustrates how a multi-interrupt system configuration might look in the IBM 1800 Data Acquisition and Control System for a typical process control application. The example serves to convey some of the principles noted above: it should not be taken as a model.

The machine configuration chosen for this example includes:

1	IBM 1802 Processor - Controller
16K	words of core storage
1	IBM 2310 Disk Storage Unit with three disk drives
1	IBM 2401 Magnetic Tape Unit
4	IBM 1053 Printer Units
1	IBM 1443 Printer Unit
1	IBM 1442 Card Read Punch Unit
1	IBM 1627 Plotter Unit
1	Analog Input Basic with Comparator
1	Analog Input Extended with Comparator
1	Digital Input
1	Digital and Analog Output
12	Interrupt levels

Other considerations are:

57 Process Interrupts (spread over 12 levels)
24 Count Servicing Subroutines
12 Programmed Interrupts
3 Timers
Queue Table size = 50

A group of process interrupts is assigned to each of 12 levels, 0-11. Note that process interrupts are normally factory wired to terminals in

INTERRUPT LEVEL STATUS WORD

INTERRUPT LEVEL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	PISW 1	TIMERS A,B,C														
1	PISW 2	2310/ 1	2310/ 2	2310/ 3												
2	PISW 3	2401														
3	PISW 4	AIBC	AIEC													
4	PISW 5	AIB	AIE	DI	DAO											
5	PISW 6	1053/ 1	1053/ 2	1053/ 3	1053/ 4											
6	PISW 7	1443														
7	PISW 8	1442														
8	PISW 9	C.I.	1627													
9	PISW 10															
10	PISW 11															
11	PISW 12															
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																
23																

NOTE

1. Interval Timers must be on a higher interrupt level than the 2310, 1816/1053, 1442 and 1443 devices.
2. The 1816/1053s must be on a lower interrupt level than the 2310s.

Figure 68. Example of Interrupt Level Status Word Assignment

groups of 4 to corresponding bit positions of one or more PISW's. In this example, only 1 process interrupt is utilized on each of the levels 0-8, and 16 process interrupts on each of levels 9-11, giving a total of 57 process interrupts.

The three timers A, B, and C are assigned to the highest interrupt level (level 0) in order to give them high response. They are thus placed to interrupt any event or device in progress. With the timers at this level, the timer servicing routines should not be calling any I/O device, but should make use of the programmed interrupt or queueing a mainline technique for servicing requirements. The reason for this is that it is not possible to call an I/O device from a level of higher priority than the I/O device being called (as already explained). In addition, it is not desirable to remain on the timer level for a period of time long enough to cause the system to miss a timed-out interval of higher priority than the one being processed.

Disk drives are assigned the next highest level (level 1) in order that the sector-gap can be made; that is, the disks can then operate at their maximum speed without incurring the penalty of a full revolution of disk time (40 ms.). The magnetic tape unit is placed at the next level (level 2) also for the same ability to service it at full capacity.

The Analog Input Basic with Comparator (AIBC) and Analog Input Extended with Comparator (AIEC) are recommended to be assigned a higher level (level 3) than their corresponding Analog Input Basic (AIB) and Analog Input Extended (AIE) devices (level 4). They must always be assigned to a higher level than the 1053 Printers.

The 1053 Printers are placed on level 5 because they are continually active, but do not require much execution time. Thus, assigning them above the process interrupts give these devices the ability to print while user's core loads are in execution.

The remaining devices present no real demand problems, and are assigned to lower levels as shown in Figure 68.

Summary of Interrupt Assignment Restrictions

For proper operation of the TSX system, the following interrupt assignment restrictions must be observed:

1. All I/O device interrupts must be assigned to a higher priority interrupt level than external interrupts, unless the external interrupt is serviced by a skeleton interrupt routine.
2. If external interrupts and I/O devices are both assigned to the same level, the external interrupts must be serviced by skeleton interrupt routines.
3. A skeleton interrupt routine cannot use an I/O device whose interrupt is assigned to the same or a lower priority level, except for the disk, 1053 Printer, and 1443 Printer; however, the 1053 test function cannot be used.
4. ILSW bits must be assigned contiguously, beginning with position 0.

Interrupts Per Level

It has been noted that a level of interrupt represents an order of precedence or priority, and that each level contains a total of 16 request positions.

When one or more lines are connected to any one priority level, it is necessary by programming means to identify the specific condition which caused that interrupt level to request service. To do this, a 16-bit word called the Interrupt Level Status Word (ILSW) is used. The programmer does not specify the ILSW in his instructions; this specification is fixed. That is, one ILSW is hardware assigned to each of 24 interrupt levels. Through the ILSW, the operational status or condition of an I/O device or process is revealed to the executive system.

The choice of interrupting I/O devices and/or process conditions on a specific interrupt level is specified by the user on the NB00-23 equate cards. If, for example, one process interrupt and four I/O devices are assigned contiguously (starting from bit zero) to level 10, the user equates NB10 to 5. The NB label is always equated to a numerical value equal to the rightmost bit (on the ILSW) plus 1 for a level. If no devices or process interrupts are assigned to a level, the label is equated to zero. Note that the NB00-23 equate cards must reflect exactly the number of bits on the System Loader assignment cards. Also, those levels that use programmed interrupts only do not contain ILSW bits; the NB cards for these levels are, therefore, equated to zero.

Level Work Areas

Whenever an interrupt caused by an I/O, a process interrupt, or a programmed interrupt occurs, an indirect branch takes place to a fixed word in core. This word contains the address of a communications area known as a Level Work Area. There is one work area per level of interrupt specified by the user, and only those levels configurated by the user will be assembled and will be available when the System Skeleton is built. In addition, three additional work areas are always assigned to the system: one each for Nonprocess Core Loads (if time-sharing is used), Mainline (that is, Process) Core Loads, and Errors (Trace and C.I. Interrupt). If time-sharing is not used, the Nonprocess work area is origined out at System Director assembly time.

Briefly, the level work area serves as a means of communications whenever the computer transfers control from one level of interrupt to another. The address of the interrupted level is then saved and the address of the work area for the current level set up. When the level in progress has completed its processing, the address of the interrupted level is restored. This method of coding automatically saves all reentrant coded subroutine work areas.

Figure 69 illustrates the layout of a level work area which is 104 words in length, but this may be increased or decreased by the user (at System Director assembly time), depending on the functions related to each level of interrupt. Note that for proper testing of errors and time-sharing, the MIC work area portion should not be deleted.

Work levels are divided into two major sections: a fixed section and an overlay section. Each word in the fixed section is assigned to one specific program and may be core protected. The manner in which it is assigned is shown in Table 11.

Words in the overlay section may be assigned to several different programs providing these programs do not call one another. This section cannot be core protected.

The overlay section begins with the 58th word of a level work area (see Table 12). If this section is used, the user is advised to reference this "start" position with a label, thus eliminating extensive program modification in the event the fixed section has increased or decreased. If a modification is required, the first word of the overlay section (the 58th word) will always be an even core location to facilitate double load and store instructions, etc. Also, programs using the overlay section should always commence at the beginning, except those programs that are called by a program already using this area. For example, DP I/O programs use the first 25 words of the overlay section; therefore any program that requires storage locations in this section and also calls a DP I/O routine cannot use the first 25 words. The later program will thus start at the 26th or 27th word of the section.

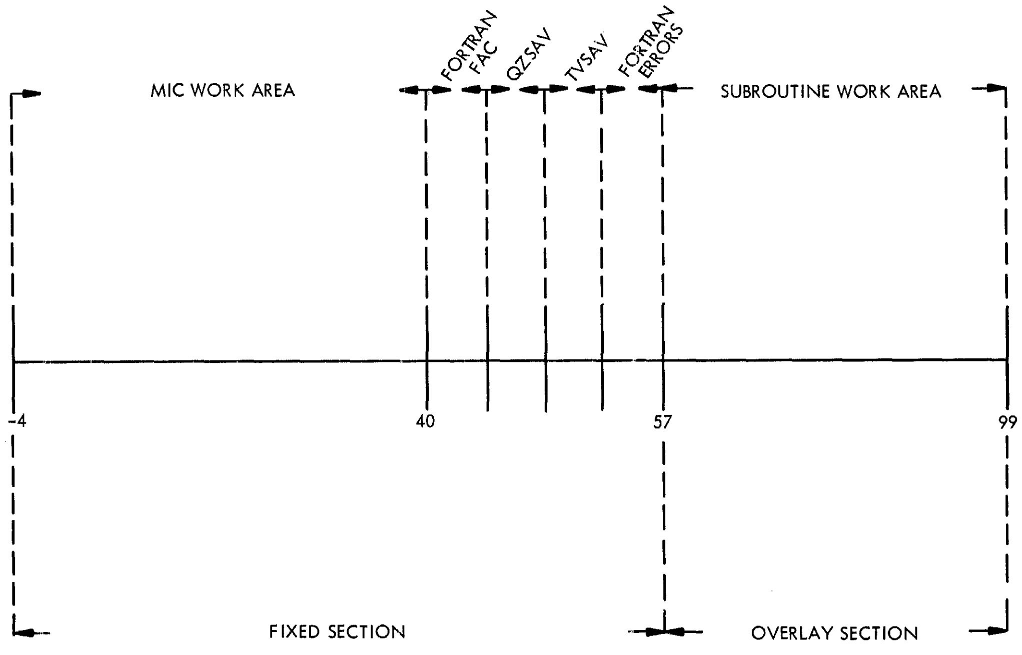


Figure 69. Layout of a Level Work Area

Table 11. Fixed Section of a Level Work Area

WORD POSITION	PROTECTION STATUS	CONTENTS
-4	N	Save interrupt exit
-3	N	Not used
-2	N	Busy indicator address for UFIO: indicates that unformatted I/O buffer has been saved to disk and must be restored by MIC.
-1	N	UFIO restore indicator; non-zero if a restore is needed.
0	Y	Address of word 5 of the ICLT entry for this level.
1	Y	Address of word 1 (in-core-with-skeleton indicator word) of the ICLT entry for this level.
2	Y	Address of word 2 (in-core-with-mainline indicator word) of the ICLT entry for this level.
3	Y	Address of word 3 (record indicator word) of the ICLT entry for this level.
4	Y	Address of word 4 (recorded indicator word) of the ICLT entry for this level.
5	Y	Address of word 0 of this level work area
6	N	Status save location
7	Y	Level number
8	N	Entry point to interrupt level coding
9	Y	STX sets this level busy
10	Y	STX saves XR3
11	Y	Sets XR3 to work level
12	Y	
13	Y	Saves XR2
14	Y	Saves XR1
15	Y	BSC long to MIC
16	Y	
17	Y	BSC long indirect to interrupt via the Master Branch Table.
18	Y	
19	N	MFIO restore indicator; non-zero if a restore is needed
20	N	Busy indicator address for MFIO; also first word of PISW IOCC.
21	N	Second word of PISW IOCC. This is a standard PISW set-up by TSX for this level. If the user wishes to sense a PISW other than the standard for this level, Word 21 would have to be modified by him. See SYSTEM DESIGN CONSIDERATIONS, SYSTEM DIRECTOR: PISW Assignment Restrictions.
22	N	Save location for XR1
23	N	Save location for XR2
24	N	Save location for XR3
25	N	Level busy indicator. Positive, if the level is busy; zero if not.
26	N	Save location for PISW sense
27	N	Save location for the address of the work level in use at the time the interrupt occurred; i.e., word 68 $_{16}$ of the Fixed Area in core.
28	N	Save location for the A-register.
29	N	Save location for the Q-register
30	N	Save location for WK4, word 36_{16} of the Fixed Area in core.
31	N	Save location for WK5, word 37_{16} of the Fixed Area in core.
32	N	CARDN indicator. If zero, detection of a // card causes an error.
33	N	Save location for MDFIO sector address on a save operation
34	N	Busy indicator address for MDFIO
35	N	File protect indicator; this must be set non-zero prior to every write to a file-protected area.
36	N	First word of ICLT (in-core skeleton address)
37	N	Second word of ICLT (in-core mainline address)
38	N	Third word of ICLT (RECORD address)
39	N	Fourth word of ICLT (RECORDED address)
40	N	Fifth word of ICLT (first entry). Words 36-40 constitute the work area used by MIC. This area is loaded with addresses from the ICLT entry for this level to inform the ALLGO routine how to handle the interrupt and/or where to find the servicing routine or core load.
41-43	N	Save locations for FORTRAN FAC (floating accumulator)
44-49	N	Locations used by QZSAV/QZEXT to save and restore: A-register Q-register XR1 XR2 Carry and Overflow indicators XR3
50-54	N	Locations used by TVSAV/TVEXT to save and restore: A-register Q-register XR1 XR2 Carry and Overflow indicators
55	N	FORTRAN functional error indicator
56	N	FORTRAN divide check indicator
57	N	FORTRAN overflow indicator

Table 12. TSX Reentrant Subroutine Work Level Requirements

SUBROUTINES	NOT REENTRANT	TVSAV	QZSAV	A-REG.	MASKED	FIXED	OVERLAY
FADD, FADDX, FSBR, FSBRX, FSUB, FSUBX		X			X	41-43, 52	58, 59, 64-66
FALOG, FLN			X		X	41-43, 55	68, 69, 72
FATAN, FATN			X		X	41-43	68, 69, 70-71, 72-73, 74
FAVL, FABS			X			41-43	
FAXB, FAXBX			X			41-43, 46, 48, 49, 55	73-79
FAXI, FAXIX		X				42, 52, 55	67, 70-72
FDIV, FDIVX, FDVR, FDVRX		X				41-43, 52, 56	58, 59, 64-66
FEXP, FXPN			X			41-43	60-63
FLD, FLDX, FSTO, FSTOX		X				41-43, 52	58, 89, 90
FMPY, FMPYX		X				41-44, 52	58, 59, 64, 65
FSIN, FSINE, FCOS, FCOSN			X		X	41-43, 55	68, 69
FSQRT, FSQR			X			41-43, 57	72
FTANH, FTNH			X			41-43, 46, 48, 49	68-73
FTRTN, FTNTR					X		80-84
EADD, EADDX, ESBR, ESBRX, ESUB, ESUBX		X			X	41-43, 52	58, 65-67
EALOG, ELN			X		X		
EATAN, EATN			X		X		
EAVL, EABS			X				
EAXB, EAXBX			X				
EAXI, EAXIX		X				42, 52, 55	69, 72, 96, 97, 99
EDIV, EDIVX, EDVR, EDVRX		X				41-43, 52, 56	76-80, 85
EEXP, EXPN			X		X	41-43	
ELD, ELDX, ESTO, ESTOX		X			X	41-43	83-84
EMPY, EMPYX		X				41-43, 52	64-66
ESIN, ESINE, ECOS, ECOSN			X		X	NONE	
ESQRT, ESQR			X			NONE	
ETANH, ETNH			X			NONE	
ETRTN, ETNTR					X	52	92-97
ADRCK					X	7	
COMGG, COMGI					X		59-63
DATSW					X	NONE	
DVCHK			X			56	
ESIGN (EXTENDED PRECISION)				X		41-43	70-75
FSIGN (STANDARD PRECISION)					X	41-43	70-75
FCTST			X			55	
IOU					X		59-63.
ISIGN					X	NONE	
ISTOX		X				50, 52	
LDFAC, STFAC, SBFAC, DVFAC					X	42	
MDFIO, MDAF, MDAI, MDCOM, MDF, MDFX, MDI, MDIX, MDRED, MDWRT						41-43	70-92
MDFND					X		72-75, 77
MFIO, MRED, MWRT, MCOMP, MIOAF, MIOIX, MIOAI, MIOI, MIOFX, MIOF						19-20, 41-43, 55	
MGOTO, MFIF, MIIF, MEIF				X	X		70-72, 74-78
MIAR, MIARX, MFAR, MFARX, MEAR, MEARX					X		70, 71, 74-85, 89
OVERF			X			57	
PAUSE		X				NONE	
SAVE, IOFIX					X	-1, 7, 11, 19, 33, 34	70-73, 88, 93, 97
SLITE, SLITT			X			NONE	
SSWTC					X	NONE	
STOP				X		NONE	

Table 12. TSX Reentrant Subroutine Work Level Requirements

SUBROUTINES	NOT REENTRANT	TVSAV	QZSAV	A-REG.	MASKED	FIXED	OVERLAY
SUBIN					X		58-83
SUBSC					X	NONE	
TSTOP					X	NONE	
TSTRT					X	NONE	
TTEST, TSET				X		NONE	
UFIO, UFIOX, UIOIX, UCOMP, UIOI, UIOF, UIOAI, UIOAF					X	-1, -2, 55	
TRACE (TRPNT)						NONE	
FARC					X	41-43, 57	
FBTD, FDTB			X			41-43	58-89
FLOAT		X				41-43, 50, 52, 53	59-62
FIXI, FIXIX		X			X	50-52, 55	
IABS					X	NONE	
IAND						NONE	
IEOR						NONE	
IFIX		X			X	41, 42, 50, 55	
IOR						NONE	
LD						NONE	
NORM		X				41-43	58
SNR					X	42, 43	
XDD		X				42, 43, 50-54	66-75
XMD		X				42, 43, 50, 51	58-65
XMDS					X	42-44	58-59
XSQR			X			44, 48	58, 59
DMPHX, DMP, DMPDC	X					NONE	
DMPS, DMPST	X					41-43	
DPART	X					7	
ENDTS				X		NONE	
LEVEL					X	NONE	
MASK					X	NONE	
OPMON						NONE	
QIFON			X		X	44-49	66, 67, 69-75, 77-84, 86-94
QUEUE			X		X		58, 61-63, 65-67
RESMK					X	NONE	
SAVMK					X	NONE	
SETCL				X		NONE	
TIMER					X	NONE	
UNMK					X	NONE	
UNQ			X		X		58, 61-63, 66, 67
VIAQ	X				X	NONE	
COUNT					X	NONE	
CLOCK				X		NONE	
CLEAR			X		X		58, 61-67
CONHX	X					NONE	
TRPRT	X					NONE	
FLIP	X					7	

(Continued)

Table 12. TSX Reentrant Subroutine Work Level Requirements

SUBROUTINES	NOT REENTRANT	TVSAV	QZSAV	A-REG.	MASKED	FIXED	OVERLAY
CARDN		X			X	32	
PAPTN		X				50-54	
MAGT		X				NONE	
PLOTX		X			X		65
REWIND/BCKSP/EOF				X		-2	66-68, 71-73, 75, 79, 85, 89
DAOP		X			X		70-75
AIPTN, AIPN		X			X		70, 71
AISQN, AISN		X			X		70-72, 74
AIRN		X			X		76-78
DIEXP		X			X		70-73
DICMP						IS REENTRANT	
DINP		X			X	50	70-72, 74, 75
ANINT, COMP1, AINT1, COMP2, AINT2	X					NONE	
AIP			X				58, 61-66
AIS			X				58, 61-71
AIR			X				58, 61-69
CO, DO, PO, DAC			X				58, 61-65
CS, VX, PI, DI			X				58, 61-65
CSC, VSC, PIC, DIC						IS REENTRANT	
CSX, VSX, PIX, DIX			X				58, 62-65
IOPE, OUSLY, ETS					X	7	
XSAVE, XEXIT, XLOAD					X	7	
GAGED, UNGAG					X	54	
QZERQ	X					NONE	
QZ010	X						58-66
BT1BT	X				X	NONE	
BT2BT	X					NONE	
BINDC		X				50	61, 62
DCBIN		X				50, 55	61
BINHX		X				50	61
HXBIN		X				50, 55	61
HOLEB		X				55	61-65
HOLPR		X				55	61-66
EBPRT		X				55	61-66
PAPEB		X				55	58-67
PAPHL		X				55	58-66
PAPPR		X				55	58-66
EBPA	X					NONE	
PRT	X					NONE	
FCHAR	X					NONE	
SCALF	X					NONE	
FGRID	X					NONE	
FPLOT	X					NONE	
ECHAR	X					NONE	
SCALE					X	NONE	
EGRID	X					NONE	
EPLOT	X					NONE	
POINT					X	NONE	
FCHRX, FCHRI, WCHRI	X					NONE	
FRULE, FMOVE, FINC	X					NONE	
ECHRX, ECHRI, VCHRI	X					NONE	

(Continued)

Table 12. TSX Reentrant Subroutine Work Level Requirements

SUBROUTINES	NOT REENTRANT	TVSAV	QZSAV	A-REG.	MASKED	FIXED	OVERLAY
ERULE, EMOVE, EINC	X					NONE	
XYPLT	X					NONE	
PLOTI, PLOTS	X					NONE	
SKELETON I/O							
DISKN			X	X		35	58-69
PRNTN		X					70-80
TYPEN/WRTYN		X					70-80

(Concluded)

Table 12 illustrates the work level requirements of TSX reentrant subroutines. These may depend on various modification levels of the TSX system. If absolute information is required, the current listings should be referred to.

The five status columns (NOT REENTRANT, TVSAV, QZSAV, A-REG, and MASKED) indicate whether each subroutine is reentrant, and if it is, what modes of reentry are used. For example, FADD (Floating-point ADD) is reentrant since the first column is blank; it uses TVSAV, but note that it also masks all levels at one or more points within the subroutine.

Some subroutines are reentrant, but do not use any words in the level work area. ENDTS is such an example. VIAQ is not reentrant; it does, however, mask all levels to prevent the Queue Table from being modified by QUEUE, QIFON, and UNQ during several instructions.

Level work areas are defined by the USE labels USE00-23), the number of work areas being determined by NULEV. If a USE label is equated to 1, a work level is included on that level; if zero, no work level is included. For example, if NULEV = 7, USE00-06 are all equated to 1; the remaining USE cards being equated to zero.

See also: Programming Subroutines Using Reentrant Coding.

Process Interrupts Per Level

Like the ILSW, the Process Interrupt Status Word (PISW) is a 16-bit word associated with the use of process interrupts. Process interrupts are physically terminated on 16-position terminal blocks within the 1800 system. The PISW indicators are turned on or off by contact closures or voltage shifts in the process. A total of 24 PISWs are allowed in the system for normal usage. To provide the maximum number of interrupt levels for process interrupts, one PISW could be assigned to each ILSW. For multiple groups per level, see PISW Assignment Restrictions.

The System Director must also be aware of the number of process interrupt bits on each of the 24 hardware levels. This information is provided by the user on the NIL00-23 equate cards. If, for example, one process interrupt is assigned to interrupt level 10, the bit configuration for the PISW for that level could be bits 0 to 15. For example, if bits 0 to 7 are assigned, the NIL10 label will be equated to 8. The NIL label will always be equated to a numerical value equal to the rightmost PISW process bit position used plus one. If no process interrupts are assigned to a level, the label will be equated to zero.

With this information, an interrupt core load table (ICLT) is built which contains an entry for each interrupt level assigned by the user plus two entries for programmed interrupts and two entries for count routines. The user specifies how many process interrupts he has on a particular level and only those words that are necessary to contain his configuration are entered in the table.

Figure 70 illustrates a partial ICL table, for one level, say level seven.

IN SKELETON represents the PISW bits of the PISW associated with the level this entry serves. If a subroutine is loaded as part of the skeleton to serve a process interrupt, a bit is set up in this word which corresponds to the bit on the PISW. For example, if the user has a process interrupt on level 0, and this PISW is wired such that when the interrupt occurs the zero bit comes on, a bit is put into the corresponding zero bit of the IN SKELETON word of the level zero ICLT entry.

The start address of the servicing routine is then loaded into START ADDRESS.

IN MAINLINE CORELOAD and RECORD are set up each time a mainline core load is read into core. The former word specifies that the interrupt servicing

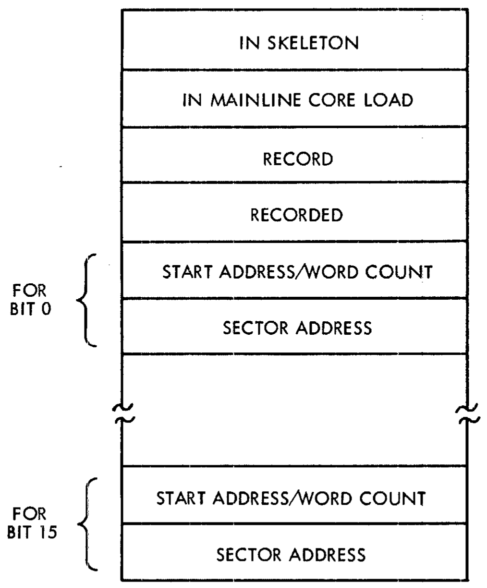


Figure 70. Interrupt Core Load Table

routine is in with the core load. RECORD means that the interrupt is not to be serviced, but only an indication that it has occurred is to be set.

RECORDED is used whenever an interrupt has been specified to be recorded during the processing of core loads. In such an event, if the interrupt occurs, the corresponding bit is set on by MIC and the interrupt turned off. These indicators are reset by the CALL QIFON and CALL CLEAR subroutines when called by the user.

The first four words of the ICL table are fixed, two additional words being required for each bit of the PISW used (see Figure 70). ICL table size is dictated by the NIL equate cards. In the example given (see Figure 66), 57 process interrupts were used; this required 258 words. If the maximum possible number of process interrupts (384) were utilized, 768 plus 4 (multiplied by the number of levels used) words of storage would be required.

If the interrupt routine were not in the skeleton, START ADDRESS would contain the word count of the routine on disk to service the interrupt. SECTOR ADDRESS would then contain the sector address of this out-of-core interrupt core load.

It should be noted that a programmed interrupt does not make use of a PISW bit for operational indication. The indicator which specifies that a programmed interrupt has occurred is set-up by the user's routine in core when he does a Call Level (see Programmed Interrupts).

PISW Assignment Restrictions

PISW (Process Interrupt Status Word) groups can be assigned to interrupt levels either as a single group per level or in multiple groups per level. For proper operation of the TSX system, the following rules and restrictions must be observed.

One Group Per Level

Normal usage of process interrupts requires that only one group of process interrupts be assigned to each interrupt level. Process interrupts assigned in this way can each be serviced with separate interrupt routines. The servicing routines must reside in the skeleton area only if their associated interrupt level is equal to or higher than any I/O device interrupt level.

When only one PISW is connected to a level, the correlation of the interrupt level number to the PISW group number is as follows:

Interrupt Level	PISW Group	Second Word of IOCC for PISW Sensing
0	1	/5F02
1	2	/5F03
2	3	/5F04
3	4	/5F05
●	●	●
●	●	●
●	●	●
22	23	/5F18
23	24	/5F19

Note that MIC performs the ILSW and PISW sensing, and transfers control to the proper interrupt servicing routine.

Multiple Groups Per Level

In special cases, such as (1) when fast response is desired, and (2) when each bit does not require a unique program to service it (such as when all the on-bits in a group might represent a particular code), it is desirable to have more than one PISW group assigned to an interrupt level; however, the following restrictions must be observed by the user's routine. The interrupt servicing (ISS) subroutine must:

1. Reside within the skeleton area
2. Sense all PISWs assigned to the level

3. Upon completion, exit to MIC via the I/O exit (that is, BSI I 90).

When assigned in this way, there is no correlation restriction between the interrupt level number and the PISW group number.

Combination PISW Assignments

It is also possible to combine the two assignment methods and have some interrupt levels with only one PISW each, and other levels with more than one PISW. The same rules and restrictions for each type outlined above still apply. For example, to have two groups of four PISWs each assigned to interrupt levels 4 and 5, one valid combination is:

Interrupt Level	PISW Group	Second Word of IOCC for PISW Sensing
0	1	/5F02
1	2	/5F03
2	3	/5F04
3	4	/5F05
4	One or more groups of PISWs	User-Sensed
5	One or more groups of PISWs	User-Sensed
6	7	/5F08
7	8	/5F09
•	•	•
•	•	•
•	•	•
17	18	/5F13
18		
19	Not assignable; usage assumed on levels 4 and 5	
•		
•		
•		
23		

Any combination can be used for the PISW assignments on levels 4 and 5.

Note that the user is not restricted in assigning multiple PISWs only to those levels which are not sensed by MIC: they can be assigned to any level. For example, if level 7 has the standard sense for a group of assigned PISWs, the user could include on that level another group of PISWs which he desires to sense himself. User-written interrupt servicing routines must be coded as an I/O ISS subroutine.

For further information on the assignment of process interrupts whose IOCCs are to be sensed by user-written subroutines, refer to the following: System Design Considerations: The IBM Nonprocess System; Programming Techniques - Writing Assembler Language Subroutines; IBM 1800 Assembler Language, (Form C26-5882).

Programmed Interrupts

CALL LEVEL -- Programmed Interrupt

External interrupt levels can also be programmed. Programmed interrupts are initiated by the CALL LEVEL statement, and follow the same basic rules which pertain to process interrupts. The format of this statement is as follows:

CALL LEVEL (L)

where

L is an integer constant 0-23 that specifies the interrupt level desired

There can only be one programmed interrupt routine per assignable interrupt level. The use of the servicing routine may, however, be expanded by setting a numeric value (integer) in Inskel COMMON prior to a CALL LEVEL (L). The LEVEL subroutine can then interrogate this value introduced by the calling program. In this way, as many subroutines per level as are desired can be provided. CALL LEVEL, which can be used only in process mainlines or interrupt programs, causes a pseudo ILSW bit to be set in the programmed interrupt IOCC (locations 160_{10} and 162_{10})in the Fixed Area. The IOCC bits are interrogated by MIC when an interrupt occurs on a specific level. MIC always reactivates those levels that have been specified within the CALL statement, and which have not been serviced.

The programmed interrupt is recognized immediately when called from a lower level. When the servicing routine finally exits to MIC, program operation at the calling level is resumed with the statement following the CALL LEVEL statement.

A programmed interrupt called from a higher level is recognized after the calling program is completed, and after any intervening interrupts are serviced. If a level is called and any ILSW bit is on when this level is recognized, the programmed interrupt is recognized after the first ILSW "on" bit is serviced.

The equate card NLSW1 is used to specify the lowest priority level number (within the group 0-13) plus 1 assigned to a programmed interrupt. Similarly, NLSW2 applies to the group 14-23. This determines the number of programmed interrupts available to the user.

In the example given in Figure 66, the number 12 punched in columns 35-36 means that programmed interrupt space in the ICL table has been allocated for levels 0-11 (12 levels). NLSW2 is equated to 0.

Consider a further example. Ten (10) programmed interrupts are required on levels 14-23; none on levels 0-13. NLSW1 is equated to 0 and NLSW2 is equated to 10.

Note that there can only be one programmed interrupt routine per interrupt level. The user need not initially have programs on disk corresponding to all levels of programmed interrupt stated. These may be loaded later.

Out-of-Core Interrupt Levels

The user has also to determine those levels on which interrupts will be serviced by out-of-core interrupt core loads. Other interrupts on levels that are serviced by out-of-core interrupt core loads are masked so that they cannot interrupt another interrupt being serviced by a corresponding out-of-core routine. Interrupts serviced by in-core routines are not masked. Only one level of exchange is maintained.

Interrupts that are in the skeleton should not be on the same or lower level as out-of-core interrupts, unless those interrupts can be masked for the period of time required to service the out-of-core interrupts.

The best practice for servicing interrupts is to group those interrupts that are serviced by in-skeleton routines on higher priority levels than interrupts which are serviced by interrupt core loads. This is not, however, a restriction on the system: in-skeleton routines may be intermixed with interrupt core loads as long as the skeleton routines can be masked for the period of time required to service the interrupt core loads.

The user specifies those interrupt levels he has elected to be masked for the servicing of interrupt core loads on two equate cards: ICLL1, ICLL2. He may elect a single interrupt level or consecutively numbered interrupt levels within the two groups of interrupt levels, 0-13 and 14-23. Two examples illustrate how the ICLL1-2 cards are used.

In the first example, assume

NULEV = 12
Three interrupt core load interrupt levels are required (9, 10, and 11).

The equated representation is shown below:

HEX EQUIV	0				0				7				F			
INT LEVEL	0	1	2	3	4	5	6	7	8	9	10	11	12	13		
ICLL1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

HEX EQUIV	F				F				F				F			
INT LEVEL	14	15	16	17	18	19	20	21	22	23						
ICLL2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Note that this corresponds to the machine configuration given in Figure 66.

In the second example, assume

NULEV = 24
Interrupt core load interrupt levels 7, 8, 9 and 19, 20 are required.

The example assumes that levels 10-18, 21-23 contain in-skeleton routines whose servicing can be delayed until the interrupt core loads have been serviced. Furthermore, if interrupt core loads on levels 7-9 are infrequently used, the normal servicing of in-skeleton interrupts (levels 10-18) would not be inhibited during the servicing of interrupt core loads on levels 19, 20.

As shown below, to obtain this, ICLL1 is equated to /01C3; ICLL2 to /063F.

HEX EQUIV	0				1				C				3			
INT LEVEL	0	1	2	3	4	5	6	7	8	9	10	11	12	13		
ICLL1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1

HEX EQUIV	0				6				3				F			
INT LEVEL	14	15	16	17	18	19	20	21	22	23						
ICLL2	0	0	0	0	0	1	1	0	0	0	1	1	1	1	1	1

NUMBER OF CALL COUNT SUBROUTINES REQUIRED BY USER

Call Count subroutines are user-written servicing routines which are assigned by *INCLD control cards to be included in the skeleton area when the skeleton is initially built. Their function is to service TSX CALL COUNT statements.

Examples of the use of count routines are:

1. Scanning variables at periodic intervals
2. Initiating control adjustments at appropriate times
3. Constructing a log

Count servicing may also be achieved by not including a subroutine. In such cases, the event is only recorded during the normal processing of core loads, and later serviced by a CALL QIFON or CALL CLEAR statement in the mainline program.

In practice, count subroutines should have short execution times and are normally only used to set an indicator, such as the setting of a programmed interrupt to some lower priority level, or queueing a mainline core load. For example, if a count routine is used to run a scan, the scan should be run from a lower priority level. This means that the count subroutine then sets a programmed interrupt: the scan routine itself could be in the skeleton, in the mainline or on the disk since programmed interrupts have the same capabilities as process interrupts. If a log is required, the log core load should be queued from the count subroutine. This means that the count subroutine calls QUEUE specifying a mainline core load. The log core load would then be executed in sequence according to the priority assigned to it. If time-sharing is used, a CALL ENDTS statement should be included within the count routine immediately following the CALL QUEUE.

The number of count routines planned by the user is punched in the NITP1 and NITP2 System Director equate cards. For example, if the user elects to use a total of 12 (Skeleton-included and recorded) count routines in his system, NITP1 and NITP2 are equated to 12 and 00, respectively. See also example in Figure 66.

During System Director assembly time, an allocation of two words per entry is set up in the Interrupt Core Load Table for the start address of each count routine, as illustrated in Figure 70.

If a count routine is included in the skeleton area, the Skeleton Builder, at skeleton build time, places its start address in the ICL table. On the other hand, if a routine is not included in the skeleton area, the numbered count only will be recorded during the normal processing of core loads. That is, a corresponding bit is "set on" by ITC in the ICL table and the servicing timer for that event is turned off. This means that the user may want to "wait" on one of the count indicators to "come on" (recorded) prior to servicing an event. This indicator is later reset by the CALL QIFON or CALL CLEAR statement when requested by the user.

The size of the ICL table is determined at assembly time from data given in the equate cards. Its entries are inserted by the Skeleton Builder and DUP. A maximum of 32 count routines is allowed.

DISK SYSTEM CONFIGURATION

Magnetic tape oriented systems are of value when applied to pure data acquisition tasks where very large storage requirements exist. It has been found, however, where control is involved, that core storage must be supplemented by a rotating memory such as a disk file to operate an efficient monitor program. The data transfer rate of 32K words to and from each disk, the shorter seek time and higher reliability of a disk file make it ideally suited for process control and data acquisition purposes.

In the IBM 1800 Time-Sharing Executive System, the software design concept is based on the use of a key unit -- a single, high-capacity disk storage unit (the IBM 2310 Disk Storage Unit) as a programming systems residence device.

Conceptually, the disk is treated in much the fashion as a reel of magnetic tape, and organized in a sequential manner. Any one segment of the disk is quickly available by moving the access mechanism directly to the starting point of a block of information, and serially transferring data. This approach also permits the user to easily store and/or retrieve blocks of data such as core loads, programs, matrices, and tables. The ability to read and write data on the same device allows the user to modify resident programs without the necessity of a second storage unit, such as in magnetic tape systems.

The disk cartridge is designed to permit rapid interchange of disks under conditions which afford adequate protection of recording surfaces. The cartridge design also ensures precision control of disk rotation at high speeds past the recording heads.

Up to three disk drives are available with the 1800 system. Typical disk uses include the following:

1. The disk file is well suited for the storage of executive programs, subroutine libraries, diagnostic routines, etc., as well as the implementation of the 1800 FORTRAN language.
2. As a storage device for recording raw data, the disk file is extremely inexpensive, and obviates the need for the more costly magnetic tape drives.
3. As a device for storing edited or corrected data, the removable disks can be transferred for permanent storage or for use on another 1800 machine for off-line computations.
4. It can be included to allow additional working storage for customer programs, or it can be used as a "spare file" for greater reliability and maintenance flexibility.

Furthermore, the disk cartridge allows each user at a system installation to "customize" standard programming packages, and to include his own subroutines and programs according to his particular needs. Tailoring resident-device characteristics to match the performance of the processor-controller makes it possible to balance the system for optimum performance.

DISK ORGANIZATION

The 2315 Disk Cartridge is organized into 200 cylinders (plus three spare cylinders) of two tracks each: one track per disk surface. For ease of block handling, each track is further divided into four sectors, each sector having a fixed length of 320 sixteen-bit words. The sector is defined as the basic addressable unit of disk storage for reading and writing.

Information is written on or read from the disk by a pair of magnetic read/write heads, one head for each surface of the disk. The three spare cylinders ensure that the stated capacities are maintained for the life of the cartridge.

Figure 71 illustrates the relationships between bits, data words, disk blocks, sectors, tracks, and cylinders for a disk storage unit.

No. Of / Per:	Word	Disk Block	Sector	Track	Cylinder	Disk
Bits	16	320	5,112	20,480	40,960	8,192,000
Data Words		20	320*	1,280	2,560	512,000
Disk Block			16	64	128	25,600
Sectors				4	8	1,600
Tracks					2	400
Cylinders						200

*These follow the first actual word of each sector, which is used for the address.

Figure 71. Disk Storage Unit Conversion Factors

Sector Numbering and File Protection

In the interest of providing disk features permitting versatile and orderly control of disk operations, two important conventions have been adopted which govern sector-numbering and file protection. Successful use of the disk subroutine, DISKN, can be expected only if user programs are built within the framework of these conventions.

The primary concern of the conventions is the safety of data recorded on the disk. To this end, the file protection scheme plays a major role, but only in a manner that is dependent upon the sector-numbering technique. The latter contributes to data safety by allowing the disk subroutine to verify the correct positioning of the access arm before it actually performs a write operation. This verification requires that sector identification be pre-recorded on each sector, and that subsequent writing to the disk be accomplished in a manner that preserves the existing identification. DISKN has been written to comply with these requirements.

Sector Numbering

Each disk sector is assigned a logical address from 0, 1, ... 1599 corresponding to the sector's position in ascending sequence of cylinder and sector numbers from cylinder zero (outermost), sector zero through cylinder 199 (innermost), sector 7. Since the disk cartridge is divided into 1600 sectors, DISKN can now address any one of these sectors.

This sector address is recorded by a standard TASK utility program (TDWA) in the sector's first word, and occupies the rightmost eleven bit positions. Of these eleven positions, the three low-order positions serve to identify the sector (0-7) within each cylinder. Utilization of this first word for identification purposes diminishes the per sector availability of data words to 320; transmission of full sectors of data is therefore performed in units of this amount.

File Protection

File protection is provided to guard against the inadvertent destruction of previously recorded data. This control can be achieved by having the normal writing functions uniformly test the file-protected status of cylinders they are about to write.

File protection is implemented in the TSX system by defining any cylinder as being file protected or not file protected. The DUP *DWRAD function is used to designate file protection. If a cylinder is file-protected, the sector address on that cylinder will contain a one-bit in bit position zero of the sector address word.

Disk Layout

The two hundred cylinders of a disk cartridge for a single disk-drive TSX system are divided into two major logical sections for system operation: a nonprocess portion and a process portion as shown in Figure 72. The nonprocess portion, known as the IBM Systems Area, is used to store the integral component parts of the IBM Nonprocess System as discussed under System Design Considerations: The IBM Nonprocess System.

Areas of disk storage contained within the process portion will now be explained in some detail. These areas are, in general, automatically assigned by the system, but they vary in size depending upon the disk system configuration and customer definition. As will be shown later, some of these areas can be modified, removed, or relocated to another storage device.

User Area. The User Area is so called because it is used to store user-written as well as IBM-furnished subroutines. It is variable in length and is file-protected. It is divided into two component areas:

1. Relocatable Subroutine Area -- an area where all relocatable IBM and user-written subroutines are stored (not shown in Figure 72).
2. Relocatable Program Area -- an area where all user-written nonprocess relocatable programs are stored as a result of a DUP *STORE control record function. This is a user-defined area -- that is, its boundary increases or decreases Nonprocess Work Storage as programs are deleted or added respectively. The number of sectors required for a program to be stored is subtracted from the Nonprocess Work Storage if they are both assigned to the same disk cartridge. When a program is deleted (by an

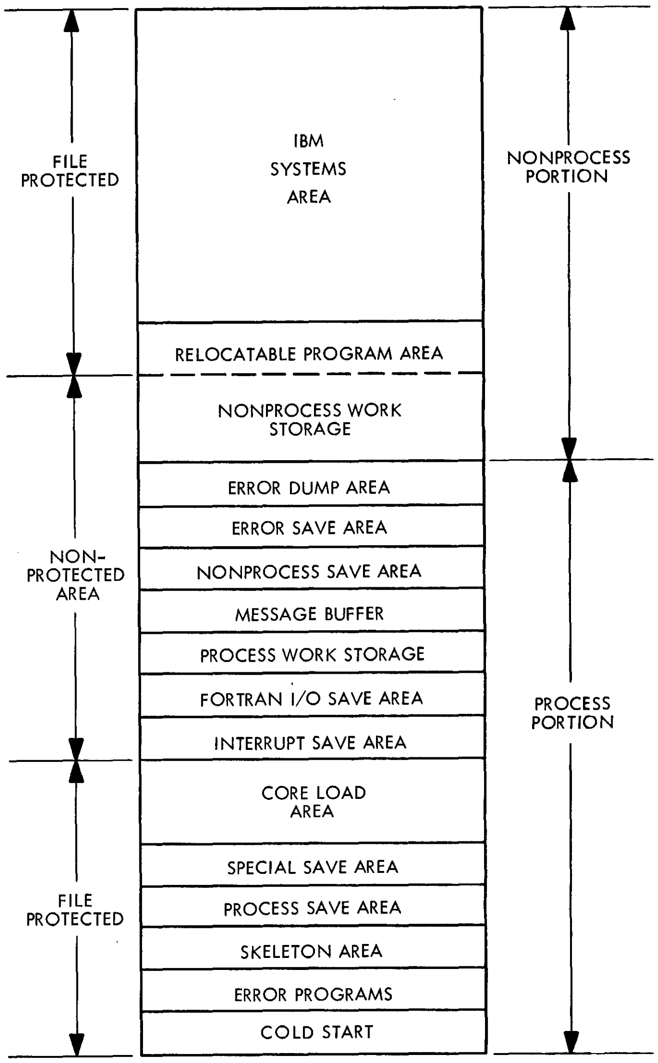


Figure 72. Disk Layout of a Single Disk Drive TSX System

*DELETE function), a separate DUP *DEFINE PAKDK operation must be performed to repack the area. The reason for this is that the relocatable program has not been overlaid and packed -- the area it occupied is thus not available for program storage. The *DEFINE PAKDK operation will cause this area to be incorporated into Nonprocess Work Storage.

The packing operation could be executed each time an *DELETE function specified a nonprocess program, but since this operation need not be performed unless a shortage of nonprocess area develops, and since the packing takes some time, it is only carried out at periodic intervals when the *DEFINE PAKDK control record is processed. The user should, however, assure himself that he has a satisfactory audit trail in the event some information is lost during the running of the disk packing operation.

Note that the area available for user programs can be expanded by using the *DEFINE REMOV function to remove the Simulator, Assembler, or FORTRAN Compiler. As an example, if all three programs are removed, the nonprocess system will be reduced to about 11 cylinders. A Relocatable Program Area can be assigned to each of up to three drives.

Nonprocess Work Storage (NPWS). This area is always adjacent to the Relocatable Program Area, and is used for temporary storage during the execution of nonprocess programs. It is used extensively during the operation of the Nonprocess Monitor. For example, it is used by the Assembler during the assembly of a program and to store the successfully assembled program in relocatable format. In a FORTRAN nonprocess program, the DEFINE FILE statement can refer to this area. The Nonprocess Work Storage is variable in length, and increases or decreases as programs are added to or deleted from the Relocatable Program Area. A NPWS area can be assigned to each of up to three disk drives.

Error Dump Area (EDPM). When a machine error occurs, the TSX Error Alert Control (EAC) program optionally writes all of core to this area on disk. The number of sectors used is directly proportional to object core size: a 32K object core requires 103 sectors, 16K requires 52 sectors, and 8K requires 26 sectors. This is a user option specified at system generation time.

Error Save Area (EPSV). This area is used when EAC is executed to save the portion of variable core used by the error detection program. An Error Save Area of six sectors is always automatically allocated when the E parameter (see The DEFINE CONFG Operation) is specified. Only one such area is required.

Nonprocess Save Area (NPSV). An area on disk used to save a partially completed nonprocess (time-shared) program that must be saved when time-sharing is terminated. A NPSV equal to variable core is always automatically assigned to the disk. No such area is required if time-sharing is inoperative.

Message Buffer (MESS). An area used to buffer 1053 messages when a WRITE to a 1053 occurs while the 1053 is busy. Its length is established at TASK assembly time (see System Design Considerations: TASK).

Process Work Storage (PRWS). Area used for temporary data storage during the execution of process programs. It is user-defined and can be specified on each of up to three disks, but only one such area can be specified in any one *DEFINE CONFG control record.

FORTRAN I/O Save Area (FIOS). Area used to save the FORTRAN I/O buffer area when a higher level interrupt uses FORTRAN I/O. The user specifies the number of interrupt levels that use FORTRAN I/O -- the system then reserves two sectors for each interrupt level plus two additional sectors if time-sharing is operative.

Interrupt Save Area (INSV). An area of disk used to save a user-specified variable area of core when an out-of-core interrupt occurs. This is user-defined and must be equal in length to the largest interrupt core load used. The area should also be large enough to include the amount of COMMON used for communications between the mainline and the subprograms in the interrupt core load.

Core Load Area. An area of disk set aside for the storage of core loads as a result of a DUP *STORECI operation. Data files are also stored here. The *FILES control record can refer to data files located in this area.

All programs and data files stored in the Core Load Area are assigned fixed disk locations. This permits the disk location of a process program to be kept with the calling program, and results in faster access to the program. A Core Load Area can be assigned to each disk.

Special Save Area (SPSV). An area of disk used to save the variable area of core when a CALL SPECL is executed and time-sharing is inoperative. When time-sharing is specified, this area is not required unless time-sharing is entered from a core load that was entered with a CALL SPECL.

Process Save Area (PRSV). Area used to store the variable area of core when a CALL SHARE or a CALL VIAQ (with an empty queue) is entered.

Skeleton Area (SKEL). Used to store a core image copy of the TSX System Skeleton. See System Design Considerations: System Skeleton.

Error Programs (EDP). This area is always required, and is used to store the disk portion of the TSX error analysis program. This program is executed when a machine error occurs.

Cold Start (CLST). This area holds the TSX Cold Start program -- which is the program that starts the TSX system into operation by loading the System Skeleton to core and transferring control to the System Director.

THE DEFINE CONFG OPERATION

At an appropriate stage in system generation, the establishment of the user's disk configuration and changes to that configuration are performed using the CONFG code in an *DEFINE operation. The definition or redefinition of some of the parameters consists of changing a stored value on disk.

In some cases, however, the results of the definition are interrelated with the user's routines, and, as a result, the Skeleton Builder must be executed after these definitions are made. This is most notably the case when the size of the System Skeleton is changed to accommodate additional programs or data areas.

Since this operation establishes the lengths of several related disk areas, and since the user may perform the redefinition several times before a final determination of the contents of the System Skeleton area is made, the DEFINE operation is designed to establish all of these areas using a small number of configuration control records.

*DEFINE CONFG Control Card

Figure 73 gives an overview of the DEFINE CONFG operation for a single drive system. The *DEFINE CONFG control card is used when the user's variable areas are established on disk. Several types of definitions may be necessary depending upon the information punched in the card.

The control card contains a field of consecutive columns which are punched with a string of one or two character codes that have special meaning in the CONFG routine. These characters are each followed by a disk drive code to specify the drive(s) where the areas are to be established.

The areas are normally taken from the highest numbered sectors of Nonprocess Work Storage, and in the same sequence as specified for that drive in the string the user punched in the CONFG card (except for M on a one-drive system).

Other fields in the card permit the user to specify lengths for areas that are not fixed by the system. These include the length estimated for the System Skeleton in core, the interrupt area utilized for out-of-core interrupts, the core load area on disk, the process work storage on disk, and the FORTRAN I/O save area on disk.

The user must execute the DEFINE CONFG operation before he builds his skeleton. If areas established by this definition turn out to be incorrect due to miscalculation of user requirements, he will probably have to re-define them, and if LSKEL is too small and cannot be increased, rebuild his skeleton since this will not be known until Skeleton build time.

Each CONFG control card can define up to 11 user-assigned disk areas. In practice, two or three control cards should be necessary to define even the most complex systems.

In referring to Figure 73, we note that the user may employ up to nine different alpha characters in any flexible order to specify a maximum of 11 user-assigned areas, each area being related to its specific disk drive. Since we are primarily concerned with a one-disk system, disk drive zero will be indicated.

Consider the general case of an *DEFINE CONFG control card with all nine possible alpha character codes punched in the first field, as shown in Figure 73.

In executing the *DEFINE CONFG operation, the define routine first establishes the legality of the control card, then scans the field starting at column 15, and interprets each alpha character in turn from left to right. Simultaneously, it establishes areas on the system disk drive

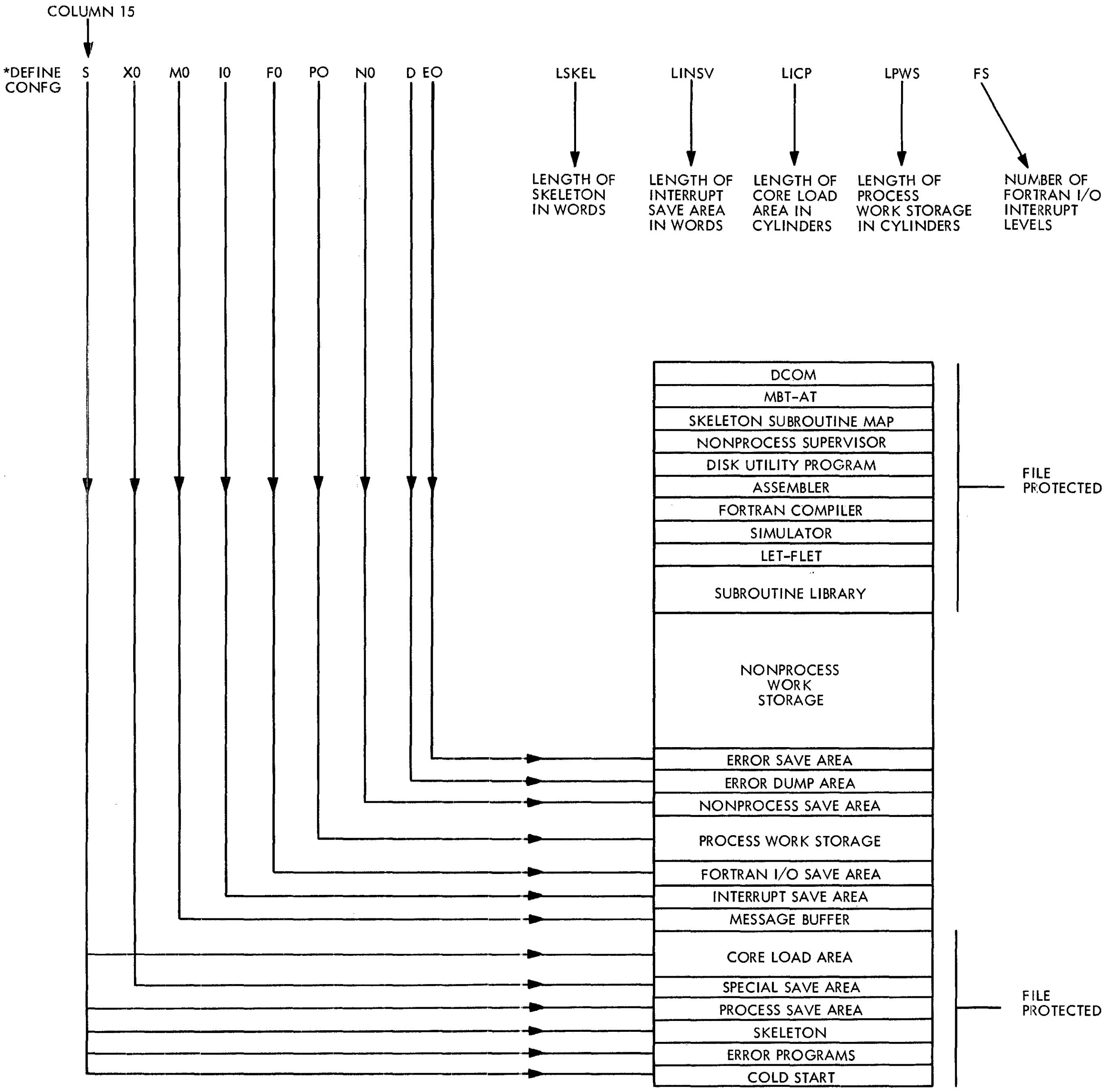


Figure 73. Overview of the DEFINE CONFG Operation (Disk Drive 0)

specified in the exact sequence of parameters punched in the card.

The disk is configurated from its highest address upwards, and definition of user-assigned areas proceeds step-by-step vertically until the scanning of the parameter columns is terminated by a blank (see Figure 74).

In initial system generation, that is, when a new disk system is first created, certain system areas are always established at the high address end of disk by specifying the S parameter in column 15. These areas constitute a basic set of programs and disk areas that are required by the system, and are file-protected at all times. They are shown below (Figure 75) in their correct sequence of allocation.

Each of these programs and areas except the Core Load Area must be defined for only one drive in the system.

Note that the Special Save Area is included within the specification of the S parameter, only if S is immediately followed by an X in column 16.

At this point, a word of explanation is given on the establishment of the Message Buffer Area (MESS). This buffer (together with the Error Programs and the Cold Start Program) is always defined at the time the IBM Nonprocess System object deck is loaded to disk by the System Loader (which computes the MESS length from data punched in the *DEDIT and TASK equate card, NOCYL). Its proper location at the high end of disk at the end of the loading operation is shown in Figure 76.

At DEFINE CONFG time, the location of MESS is redefined, that is, it is bodily moved to a different location on the disk. For example, if the F parameter immediately follows SXO in the control card, MESS will be reflected as in Figure 77.

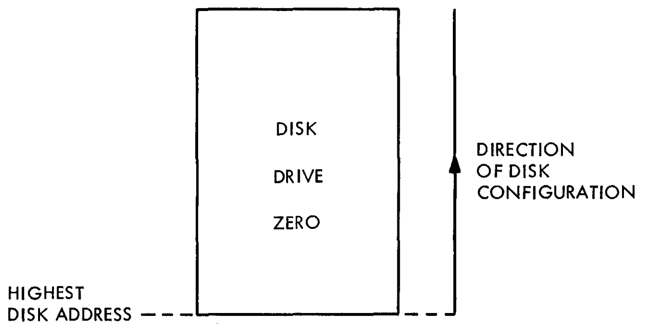


Figure 74. Illustrating Direction of Disk Configuration

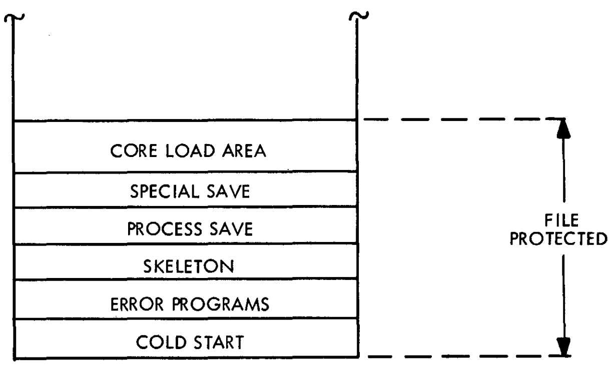


Figure 75. Establishment of System Areas at High Address End of a Disk

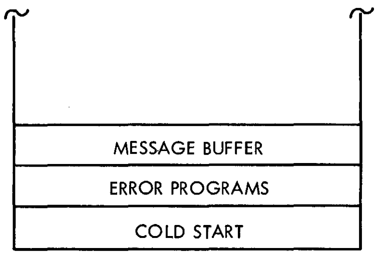


Figure 76. Establishment of Message Buffer Area at System Load Time

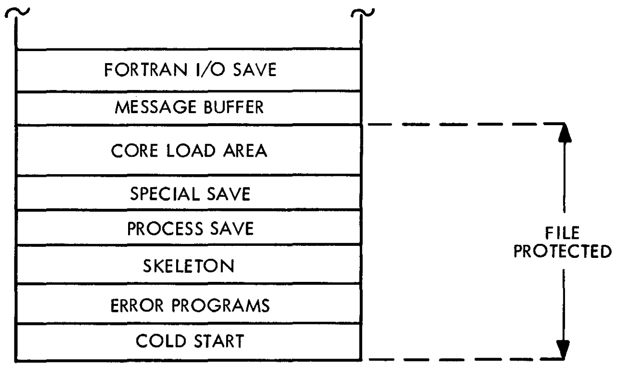


Figure 77. Illustrating Redefinition of the Message Buffer Area

The Message Buffer Area always precedes the Core Load Area on a one-drive system. Note, however, that if MESS is not required, that is, no provision is made for it in the *DEDIT control card, M should not be specified in the *DEFINE CONFG control card, and no redefinition of this area takes place.

Examples of Disk Configurations

The following examples illustrate typical disk configuration operations.

EXAMPLE 1. Define a single drive disk system such that logical disk drive zero shall contain the following areas:

- Core Load Area: 30 cylinders
- Process Work Storage: 6 cylinders
- FORTRAN I/O Save Area: 4 sectors
- Interrupt Save Area: 2000 words
- Skeleton Area: 16,000 words
- Message Buffer: 6 cylinders
- Error Dump Area
- Error Save Area
- Nonprocess Save Area

A 32K system is assumed.

It is the user's responsibility to specify the lengths of the first six areas required: the DEFINE CONFG function automatically computes and allocates the remaining three areas. Since a 32K machine is used, an Error Dump Area equal to object core size (32K) and a Nonprocess Save Area equal to variable core (16K) are assigned to the disk. An Error Save Area of six sectors is always automatically allocated when the E parameter is specified.

The control card sequence for this operation is:

SAMPLE CODING FORM

```
   1-10      11-20      21-30      31-40      41-50      51-
1234567890123456789012345678901234567890123456789012345
// JOB   X
// DUP
*DEFINE CONFG SXØIØDEØFØNØPØMØ 16ØØØ Ø2ØØØ ØØ3Ø ØØØ6 Ø2
// JOB   X
// END OF ALL JOBS
```

Figure 78 reflects the disk layout for disk drive zero on completion of this function.

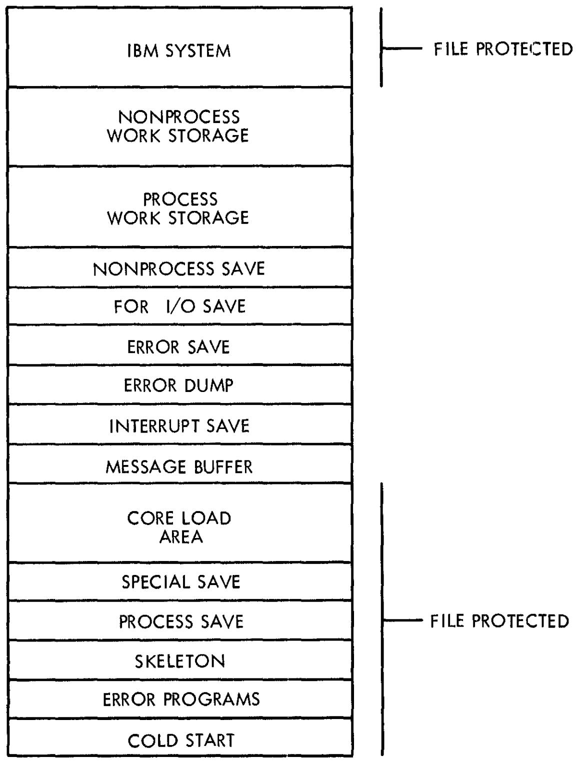


Figure 78. Disk Layout of Disk Drive Zero for Example 1

EXAMPLE 2. Define a three-drive TSX on-line system as follows:

<u>Drive Zero</u>

- Label this disk cartridge: 13579
- Core Load Area = 30 cylinders
- Process Work Storage = 5 cylinders
- FORTRAN I/O Save Area = 24 sectors
- Skeleton Area = 16384 words
- Nonprocess Save Area

<u>Drive 1</u>

- Label this disk cartridge: 12345
- Establish a LET/FLET area of 2 cylinders
- Core Load Area = 100 cylinders
- Process Work Storage = 10 cylinders
- Message Buffer = 6 cylinders

<u>Drive 2</u>

- Label this cartridge: 09876
- Establish a LET/FLET area of 1 cylinder
- Core Load Area = 125 cylinders
- Interrupt Save Area = 2048 words
- Error Dump Area
- Error Save Area

A 32K system is assumed.

As this is a multi-drive system, the total number of disk drives available to the system must be defined. Remember that the System Loader assumes at system load time that only one disk drive (logical disk drive zero) is present on the system; DCOM thus indicates only one disk drive. The *DEFINE NDISK is therefore a redefinition of the system: this must always be performed before the skeleton is built.

Disk cartridges, other than that on logical drive zero, which are intended for system usage are then initialized by the DUP *DLABL function. Since disk drives 1 and 2 have not previously been identified as system drives, they are cleared (including file-protect bits), disk addresses are written, a LET/FLET area is established as specified, and a label written on each disk cartridge. In the case of disk drive 2, a one-cylinder LET/FLET area is defined by default. Logical drive zero is relabelled with 13759.

A separate *DEFINE CONFG control card must be used for each disk cartridge if it is to contain a Core Load Area and/or Process Work Storage. Note that S is only used to specify a basic set of system programs/areas, which includes the Core Load Area, for the system drive (logical drive zero). The parameter C, indicating a Core Load Area, is only used for auxiliary disk drives (drives 1 and 2).

As pointed out in Example 1, it is the user's responsibility to specify the lengths of the following:

- Core Load Area
- Process Work Storage
- FORTRAN I/O Save Area
- Interrupt Save Area
- Skeleton Area
- Message Buffer for 1053 Printer

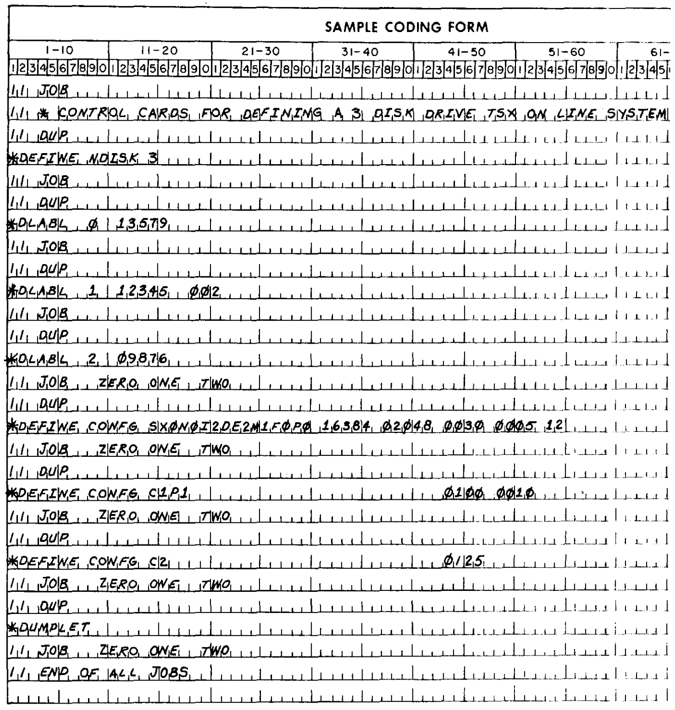
The *DEFINE CONFG function automatically computes and allocates disk storage for:

- Error Dump Area
- Error Save Area
- Nonprocess Save Area
- Nonprocess Working Storage

In our example, since a 32K machine is implied, an Error Dump Area equal to object core size (32K), and a Nonprocess Save Area equal to variable core (16K) are allocated. An Error Save Area of six sectors is automatically set aside whenever E is specified. Note that Message Buffer Size is determined by the TASK equate card NOCYL at TASK assembly time.

Figure 79 reflects the layout of the three disks following configuration.

The control record sequence for this multiple operation is given below:



SAMPLE CODING FORM

```
// JOB
// * CONTROL CARDS FOR DEFINING A 3 DISK DRIVE TSX ON LINE SYSTEM
// DUP
*DEFINE NDISK 3
// JOB
// DUP
*DLABL   Ø  13579
// JOB
// DUP
*DLABL   1  12345  ØØ2
// JOB
// DUP
*DLABL   2  Ø9876
// JOB  ZERO ONE  TWO
// DUP
*DEFINE CONFG SXØNØI2DE2M1FØPØ 16384 Ø2Ø48 ØØ3Ø ØØØ5 12
// JOB  ZERO ONE  TWO
// DUP
*DEFINE CONFG C1P1                         Ø1ØØ ØØ1Ø
// JOB  ZERO ONE  TWO
// DUP
*DEFINE CONFG C2                           Ø125
// JOB  ZERO ONE  TWO
// DUP
*DUMPLET
// JOB  ZERO ONE  TWO
// END OF ALL JOBS
```

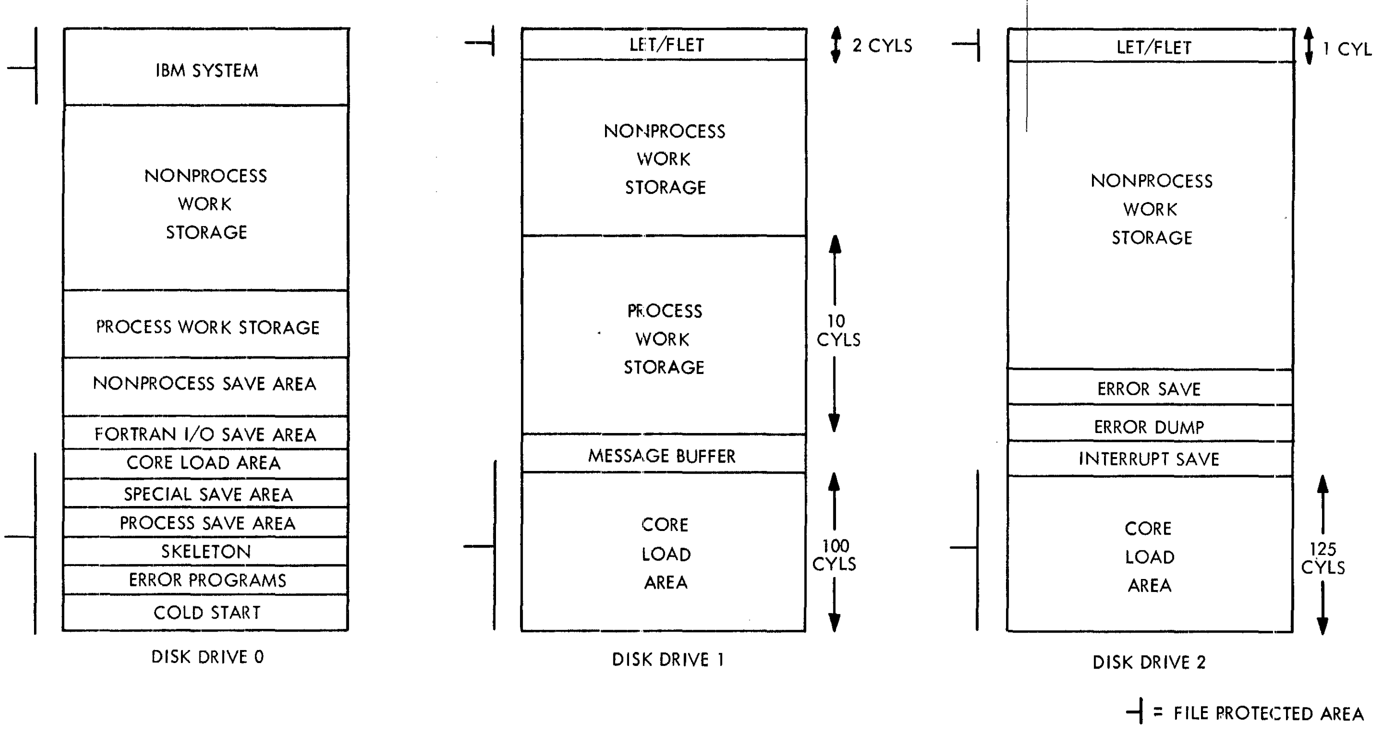


Figure 79. Definition of a Three-Drive TSX On-Line System for Example 2

EXAMPLE 3. Define a three-drive TSX off-line system as follows:

Drive Zero
- Core Load Area = 50 cylinders
- TASK Skeleton Area = 8000 words
- Message Buffer Area = say 6 cylinders
- Process Save Area

Drive 1
- Label this disk pack: 12635
- Establish a LET/FLET area of 2 cylinders
- Core Load Area = 50 cylinders

Drive 2
- Label this disk pack: 23764
- Establish a LET/FLET area of 5 cylinders
- Core Load Area = 50 cylinders

A 16K system is assumed.

The number of disk drives available to the off-line system is first defined, as in Example 2. Disk drives 1 and 2 are then initialized, and LET/FLET areas established (see Figure 80). Since each disk cartridge is to contain a Core Load Area, a separate control card is used to define each cartridge. Note that the skeleton area on logical drive zero is configurated to hold the TASK monitor system.

Figure 80 reflects the disk layouts of the off-line system.

The control record sequence is given below:

SAMPLE CODING FORM

```
// JOB
// * CONTROL CARDS FOR DEFINING A 3 DRIVE TSX OFF LINE SYSTEM
// DUP
*DEFINE NDISK 3
// JOB
// DUP
*DLABL  1 12635   ØØ5
// JOB
// DUP
*DLABL  2 23764   ØØ5
// JOB
// DUP
*DEFINE CONFG 5ØIØMØ           Ø8ØØØ       ØØ5Ø
// JOB        X    X
// DUP
*DEFINE CONFG C1                           Ø15Ø
// JOB        X    X
// DUP
*DEFINE CONFG C2                           ØØ5Ø
// JOB        X    X
// DUP
*DUMPLET
// JOB        X    X
// END OF ALL JOBS
```

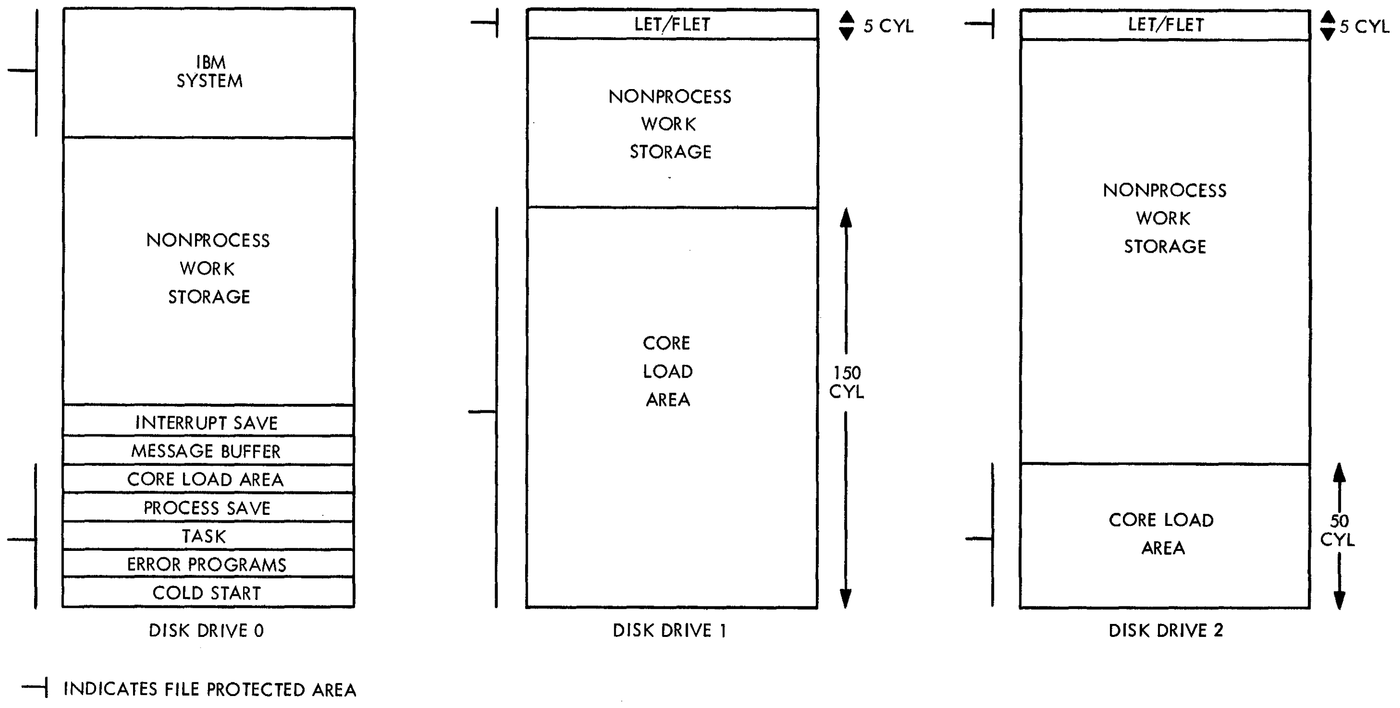


Figure 80. Definition of a Three-Drive TSX Off-Line System for Example 3

DISK CARTRIDGE INITIALIZATION

There are three programs within TSX concerned with disk cartridge initialization.

- TASK DISK WRITE ADDRESS (TDWA, a TASK utility program)
- DWRAD (a DUP function)
- DLABL (a DUP function)

A comparison of their features is given in Table 13 at the end of this section.

Use of TDWA

A disk cartridge cannot, by definition, be used for processing functions unless it is first initialized by TDWA. Thus, whenever a disk cartridge is initially supplied, or is to be re-initialized, TDWA must be used. This operation is carried out in the off-line mode under TASK control (e.g., at system generation time).

TDWA performs two basic functions: it (1) checks the ability of the disk to record and reproduce information, and (2) writes addresses on the disk, flags defective cylinders, zeroes all storage words, and records the first sector of each defective cylinder on sector zero of the disk cartridge which is file-protected. TDWA does not label the disk cartridge.

A practical example of the initialization of a two disk-drive system is given in Programming Techniques: TSX Sample System. For TASK DISK WRITE ADDRESS system procedures, see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Use of DWRAD

DWRAD allows the user to perform in an on-line or off-line mode the following functions:

- Rewrite sector addresses in any specified cylinderized area on any disk cartridge
- Retain or save the contents of the sectors indicated, if desired, for analysis purposes

- Zero from one to 199 cylinders as specified (except cylinder zero)

- Enforce file protect or file unprotect on the entire area specified

DWRAD does not label the disk cartridge.

As disk sector addresses may be inadvertently modified or destroyed during the execution of user programs, such as in the transfer of data to core by a READ command, or by hardware failures, DWRAD provides the user with the ability to rewrite sector addresses of specified areas without recourse to a re-initialization process by TDWA. Whenever addresses in a certain area are destroyed, data cannot be retained or preserved in these sectors; the area must be readdressed and the data zeroed.

The file protect/file unprotect feature is useful in those situations where the user desires to enforce or remove file protection from a specified disk file or systems area. He might, for example, require to remove file protection from a certain portion of the Core Load Area for Assembler WRITE operations, and later restore file protection to that area.

The following examples depict typical operations.

EXAMPLE 4. Two cylinders beginning at sector 408 and ending at sector 417 are to be file protected. Assume that the disk cartridge is on disk drive zero. Information on these cylinders is not to be changed.

```
                                   SAMPLE CODING FORM
   1-10      11-20      21-30      31-40      41-50
1234567890 1234567890 1234567890 1234567890 1234567890
// JOB Ø
// DUP
*DWRAD    Ø        4Ø8  417    P
// END OF ALL JOBS
BLANK CARD
BLANK CARD
```

EXAMPLE 5. Two cylinders as specified in Example 4 are to be file unprotected. Clear all sectors. Assume that disk cartridge is on disk drive zero.

```
                                   SAMPLE CODING FORM
   1-10      11-20      21-30      31-40      41-50
1234567890 1234567890 1234567890 1234567890 1234567890
// JOB Ø
// DUP
*DWRAD    Ø        4Ø8  417  Z
// END OF ALL JOBS
BLANK CARD
BLANK CARD
```

Note that file protection (or the removal of file protection) will only be effective if the disk drive indicated (in column 11) is defined to be on the system. That is, it is specified on the // JOB control card.

EXAMPLE 6. Zero cylinders 27_{10} to 32_{10} with file protection removed. Assume that disk cartridge is on drive 2.

```
                                   SAMPLE CODING FORM
   1-10      11-20      21-30      31-40      41-50
1234567890 1234567890 1234567890 1234567890 1234567890
// JOB 2           X
// DUP
*DWRAD    2        ØD8  1ØØ  Z F
// END
BLANK CARD
BLANK CARD
```

Use of DLABL

Disk cartridges on disk drives other than logical drive zero, and intended for system usage, must be initialized by means of the DUP *DLABL function.

DLABL serves three purposes:

- Places a label on the cartridge

- Establishes a LET/FLET table

- If certain conditions are met, it also writes addresses on an entire disk

DLABL places a numeric label, as specified by the user on the *DLABL control card, in word zero of sector zero on the disk cartridge. To prevent users from inadvertently destroying the system, only the label is written when the disk drive specified is drive zero, and when the drive is a system drive. All other data on these cartridges, including defective cylinder addresses in sector zero, remain unchanged.

For all other disk cartridges, DLABL assumes an unlabelled, pre-addressed disk cartridge. It then clears the cartridge, including the file-protected areas, but not including the defective cylinder addresses in sector zero of cylinder zero; writes a new label, if specified, in word one, sector zero of the disk, and establishes a file-protected LET/FLET area (LET in sector one, FLET in the last sector of the LET/FLET cylinder area). Note that unless the disk cartridge operated on is located on logical drive zero, or a system drive specified on the // JOB card, DLABL will erase all data as it writes addresses, and always establish one file-protected cylinder.

The size of the LET/FLET area is determined either by user specification on the DLABL control card, or, if unspecified, automatically made to be eight sectors.

Each DUP *DLABL function must be run as a separate job; that is, each *DLABL control card should be preceded by a // JOB, // DUP card combination, and should be followed by the next // JOB card. It can be performed in either the on-line or off-line mode.

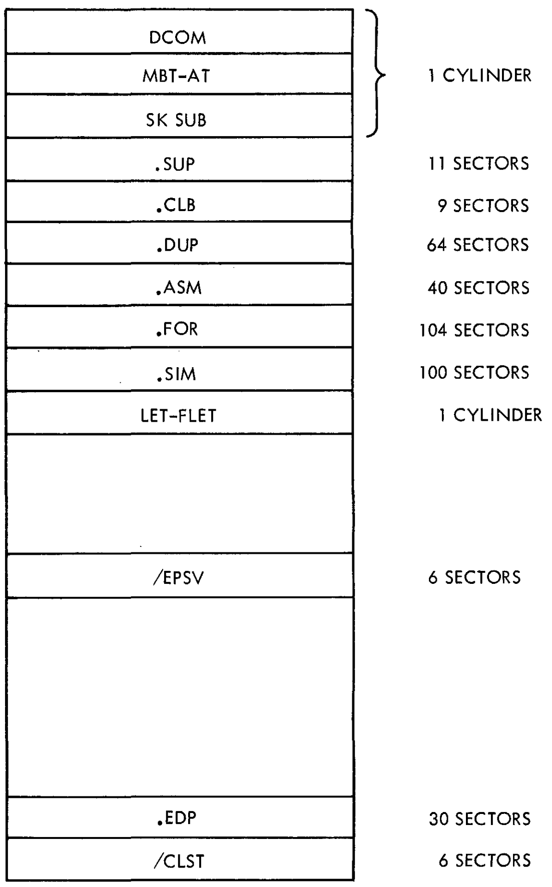
For typical DLABL operations, see Examples 2 and 3; also, Programming Techniques: TSX Sample System.

Table 13. Comparison of TDWA, DWRAD, and DLABL Features

FEATURES	TDWA	DLABL		DWRAD
		ON SYSTEM	OFF SYSTEM	
CARTRIDGE TEST	YES	NO	NO	NO
ON-LINE OR OFF-LINE	OFF-LINE	BOTH	BOTH	BOTH
CLEARS CARTRIDGE	YES	NO	YES	YES (BY SPEC. CYLS.)
FILE PROT./ FILE-UN PROT.	FILE PROT. CYL. ZERO ONLY	NO	FILE UNPROTECTS	BOTH (BY SPEC. CYLS.)
WRITES SECTOR ADDRESSES	YES	NO	NO	YES (BY SPEC. CYLS.)
WRITES LABEL	NO	YES	YES	NO
ESTABLISHES LET/FLET AREA	NO	NO	YES	NO

SUMMARY OF DISK STORAGE REQUIREMENTS AND ASSIGNMENT RESTRICTIONS

1. Disk areas that are fixed and equal for all systems, regardless of core size. These are illustrated below. Note that these areas constitute a basic nonprocess system.



2. Disk areas that are not dependent upon core size, but which may vary in disk storage requirements

   Relocatable Program Area. This will expand or contract in size as relocatable programs are added to or deleted from the system.

   Nonprocess Work Storage. NPWS will increase or decrease as relocatable programs and subroutines are added to or deleted from the system.

Message Buffer Area. The size of the Message Buffer Area is computed by the user and specified during system generation. The factors which determine its size are discussed in System Design Considerations: TASK.

Core Load Area. The size of the Core Load Area is specified by the user at DEFINE CONFG time.

System Skeleton Area. This is a copy of the skeleton area of core; its size is determined by the formula:

Skeleton Size = Total Core Size Minus Size of Variable Core

FORTRAN I/O Save Area. The number of sectors required for this area is determined by the number of interrupt levels which use FORTRAN I/O. See Disk Organization.

3. Disk areas where storage requirements depend upon the size of variable core. These include:
   - Nonprocess Save Area
   - Interrupt Save Area
   - Special Save Area
   - Process Save Area

4. Disk areas that must be assigned to logical disk drive 0:

   DCOM
   MBT-AT
   SK-SUB
   SUP
   CLB
   DUP
   ASM
   FOR
   SIM
   IBM Subroutine Library

5. Disk areas that must be assigned to one disk drive, but need not be assigned to logical disk drive zero:

   EPSV
   CLST
   INSV
   ERPG
   PRSV
   SKEL

6. Disk areas that may be assigned to one disk drive, but need not be assigned to logical disk drive zero:

   EPDM
   NPSV
   MESS
   FIOS
   SPSV

7. Disk areas that will be assigned to the same disk drive, but need not be assigned to logical disk drive zero:

   PRSV
   SKEL
   ERPG
   CLST

8. Disk areas that can be assigned to more than one disk drive:

   Relocatable Program Area
   NPWS
   PRWS
   Core Load Area

9. Disk areas that must be assigned to every disk drive:

   LET/FLET

10. Disk File Protection
    (a) Certain areas of the disk are file-protected. This means that the user cannot write into any of these areas at object time, although he can update file-protected files by using FORTRAN I/O only. System programs can, however, write into file-protected areas.
    These areas include the following:

    DCOM
    MBT-AT
    SK-SUB
    SUP
    CLB
    DUP
    ASM
    FOR
    LET/FLET
    IBM Subroutine Library
    Relocatable Program Area
    Core Load Area
    SPSV
    PRSV
    SKEL
    EDP
    CLST

(b) User-written programs are stored in the file-protected areas on disk. Programs are written into (that is, added to the system) file-protected areas by the DUP *STORE, *STOREMD, *STORECI, and *STOREDATA operations.

(c) File-protected areas are fixed in size and cannot be altered by the user.

SYSTEM SKELETON

In an on-line TSX system, a nucleus of supervisory programs and their associated work areas and tables must be permanently core-resident to obtain efficient and continuous operation. At the center of this nucleus is the System Director which provides the essential communications between core loads and interrupt servicing routines. This framework of programs is referred to as the System Skeleton.

TASK is used in conjunction with the Skeleton Builder program to construct the in-core skeleton as required for TSX system operation within the limits prescribed by the user. The size and content of the skeleton is dependent on the size of the object machine, the size of user's process programs, and the size of process core loads which the user may plan to move in and out of core during system operation. The skeleton can be considered as the permanent part of all executable core loads.

For on-line processing to take place, the System Skeleton must be loaded to core memory; this is accomplished initially by the system cold start program.

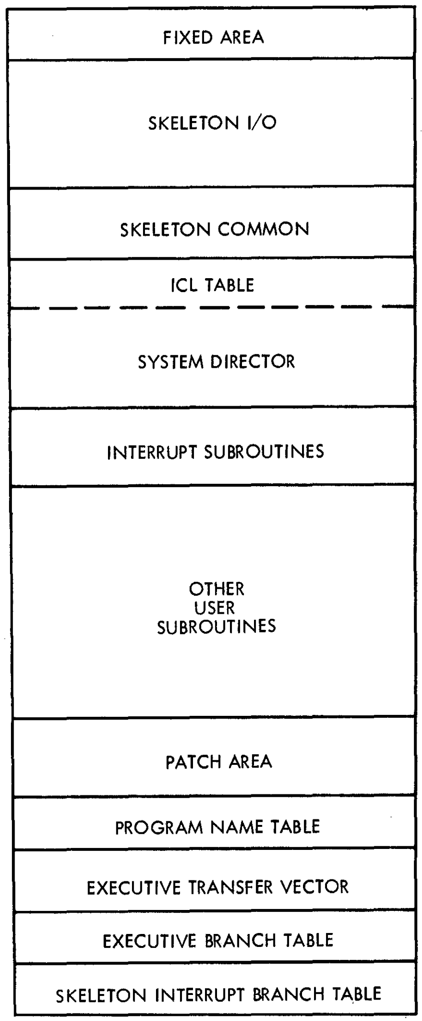


Figure 81. Constitution of the System Skeleton

CONSTITUTION OF THE SYSTEM SKELETON

Figure 81 illustrates the various component parts that make up the System Skeleton. Each of these parts and its function are explained below.

Fixed Area. This is effectively a systems communications area containing information used by all TSX system programs. It is initially assembled as part of TASK. At skeleton build time, various values in this area are initialized by the Skeleton Builder from System Director input. A disk image of the Fixed Area can be obtained by a disk dump of the first sector of the Skeleton. See Appendix C: Contents of the Fixed Area of Core.

Skeleton I/O. An identical set of input-output routines to that used by TASK forms the basis of Skeleton I/O. This permits the user to perform various disk, printer, and card utility functions (see System Design Considerations: TASK).

Skeleton Common. The maximum size of the Skeleton Common area for an object machine is defined and fixed by the user at TASK assembly time through the equate card COMSZ. COMSZ may be zero or any positive decimal value that will not cause the skeleton size to exceed the start address of variable core (that is, VCORE). TASK will determine the start address and word count of this common area and store them in words 156 and 157 in the Fixed Area. INSKEL COMMON is the only common area that is permanently core-resident. It provides communications between various core load types, and those subroutines included in the skeleton. When INSKEL COMMON is referenced in a FORTRAN program, listed variables are assigned addresses in Skeleton Common. All other attributes of the COMMON, as used in the FORTRAN language, are retained.

System Director. This forms the operating center of the TSX system. It has the responsibility of directing interrupt servicing, loading of user core loads, supervising time-sharing, and servicing of interval timers and error conditions. When the system is operating under control of the System Director, control is passed to it by TSX calls, interrupts, and error conditions. The System Director is that portion of TSX, other than the Skeleton I/O, which must be in core at all times in order to respond to a real-time environment. A detailed discussion of its functions is given in another section of this manual.

User Subroutines. The user has the option of including frequently called subroutines and high priority interrupt routines in the skeleton. These may include:

- Skeleton subroutines
- Interrupt subroutines
- Programmed Interrupt subroutines
- Count subroutines
- User-written trace and error subroutines
- Timer subroutines

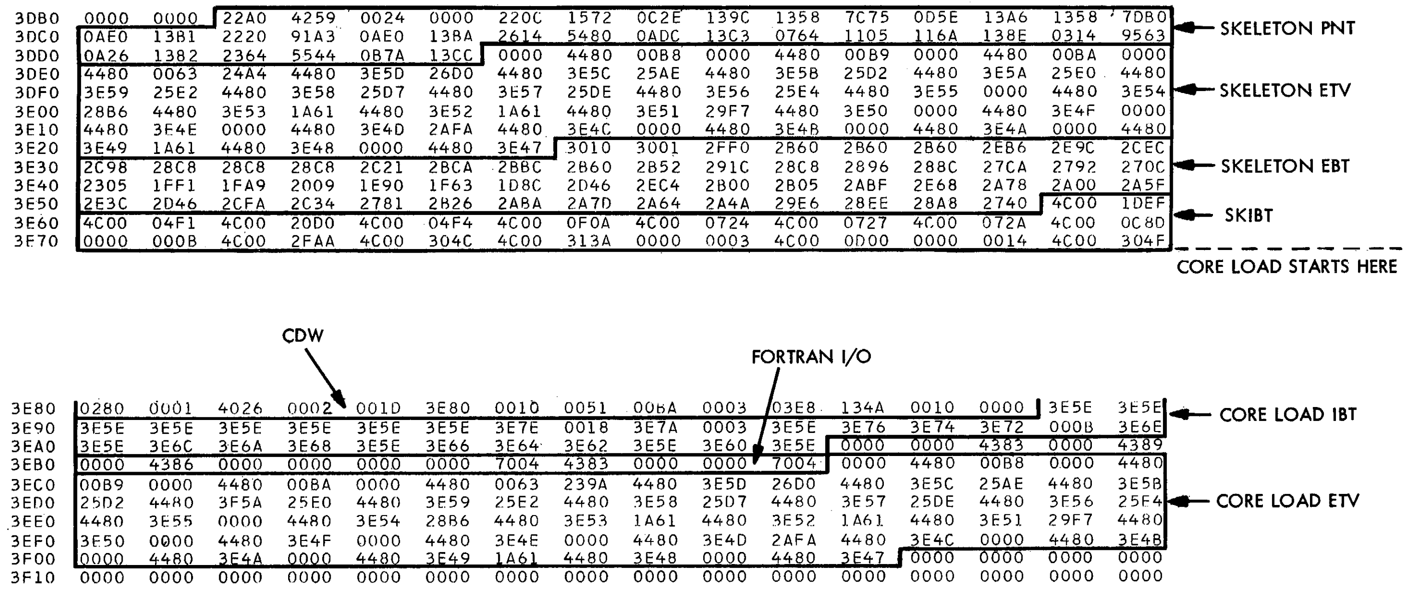
These programs must have previously been compiled/assembled by the user and stored in relocatable format in the Relocatable Program Area on disk.

In addition, if FORTRAN I/O is utilized, then those conversion routines (e.g., HOLEB, EBPRT) necessary for its proper use must also be included in the skeleton by having them specified on *INCLD control cards. A detailed examination of some of the important considerations governing the inclusion of subroutines in the skeleton is made later in this section.

Patch Area. This is the portion of core storage that remains between the end of the subroutine area and the Skeleton Program Name Table (PNT) which is allowed (not explicitly defined) for the modification of IBM and user programs within the skeleton. Its size is determined by user requirements, but should, in practice, be at least 100 words in length to allow for future IBM modifications.

Program Name Table. This is the part of the skeleton table area that maps (name, word count, and sector address) all core loads referenced to Program Sequence Control (PSC) by calls made by in-skeleton subroutines. The symbolic name (SYDIR) in truncated EBCDIC code for the System Director forms the first entry in this table. See Figure 82.

Executive Transfer Vector (ETV). The Executive Transfer Vector serves as a linkage between LIBF-type calls of a core load and corresponding routines in the skeleton. It is originally constructed by TASK; for each LIBF routine put into Skeleton I/O by TASK, an entry is made into the ETV. At skeleton build time, the Skeleton Builder inserts an entry into the ETV for each entry point of each LIBF routine placed in the skeleton, by extending the size of the original ETV to reflect the entries for the included subroutines.


SKELETON PNT
SKELETON ETV
SKELETON EBT
SKIBT
CORE LOAD STARTS HERE
CDW
FORTRAN I/O
CORE LOAD IBT
CORE LOAD ETV

```
3DB0  0000 0000 22A0 4259 0024 0000 220C 1572 0C2E 139C 1358 7C75 0D5E 13A6 1358 7DB0
3DC0  0AE0 13B1 2220 91A3 0AE0 13BA 2614 5480 0ADC 13C3 0764 1105 116A 138E 0314 9563
3DD0  0A26 1382 2364 5544 0B7A 13CC 0000 4480 00B8 0000 4480 00B9 0000 4480 00BA 0000
3DE0  4480 0063 24A4 4480 3E5D 26D0 4480 3E5C 25AE 4480 3E5B 25D2 4480 3E5A 25E0 4480
3DF0  3E59 25E2 4480 3E58 25D7 4480 3E57 25DE 4480 3E56 25E4 4480 3E55 0000 4480 3E54
3E00  28B6 4480 3E53 1A61 4480 3E52 1A61 4480 3E51 29F7 4480 3E50 0000 4480 3E4F 0000
3E10  4480 3E4E 0000 4480 3E4D 2AFA 4480 3E4C 0000 4480 3E4B 0000 4480 3E4A 0000 4480
3E20  3E49 1A61 4480 3E48 0000 4480 3E47 3010 3001 2FF0 2B60 2B60 2B60 2EB6 2E9C 2CEC
3E30  2C98 28C8 28C8 28C8 2C21 2BCA 2BBC 2B60 2B52 291C 28C8 2896 288C 27CA 2792 270C
3E40  2305 1FF1 1FA9 2009 1E90 1F63 1D8C 2D46 2EC4 2B00 2B05 2ABF 2E68 2A78 2A00 2A5F
3E50  2E3C 2D46 2CFA 2C34 2781 2B26 2ABA 2A7D 2A64 2A4A 29E6 28EE 28A8 2740 4C00 1DEF
3E60  4C00 04F1 4C00 20D0 4C00 04F4 4C00 0F0A 4C00 0724 4C00 0727 4C00 072A 4C00 0C8D
3E70  0000 000B 4C00 2FAA 4C00 304C 4C00 313A 0000 0003 4C00 0D00 0000 0014 4C00 304F
```

```
3E80  0280 0001 4026 0002 001D 3E80 0010 0051 006A 0003 03E8 134A 0010 0000 3E5E 3E5E
3E90  3E5E 3E5E 3E5E 3E5E 3E5E 3E5E 3E7E 0018 3E7A 0003 3E5E 3E76 3E74 3E72 000B 3E6E
3EA0  3E5E 3E6C 3E6A 3E68 3E5E 3E66 3E64 3E62 3E5E 3E60 3E5E 0000 0000 4383 0000 4389
3EB0  0000 4386 0000 0000 0000 0000 7004 4383 0000 0000 7004 0000 4480 00B8 0000 4480
3EC0  00B9 0000 4480 00BA 0000 4480 0063 239A 4480 3E5D 26D0 4480 3E5C 25AE 4480 3E5B
3ED0  25D2 4480 3F5A 25E0 4480 3E59 25E2 4480 3E58 25D7 4480 3E57 25DE 4480 3E56 25E4
3EE0  4480 3E55 0000 4480 3E54 28B6 4480 3E53 1A61 4480 3E52 1A61 4480 3E51 29F7 4480
3EF0  3E50 0000 4480 3E4F 0000 4480 3E4E 0000 4480 3E4D 2AFA 4480 3E4C 0000 4480 3E4B
3F00  0000 4480 3E4A 0000 4480 3E49 1A61 4480 3E48 0000 4480 3E47 0000 0000 0000 0000
3F10  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

Figure 82. A Partial Dump Following a Skeleton Build to Illustrate the Program Name Table and the Executive Transfer Vector

Each entry in the ETV is three words long, corresponding to the format of the VTV table associated with each core load. When the skeleton build process is completed, word 3 will contain the address of the corresponding entry in the Executive Branch Table (EBT).

The Executive Transfer Vector is, in reality, a copy or duplicate of the ETV in all core loads, and is effective only during disk-to-core transfers.

Figure 82 is given to provide a better insight of the Program Name Table (PNT) and the Executive Transfer Vector (ETV). It is a partial dump snapshot of the skeleton built for the 1800 TSX-Sample System (see Programming Techniques), and should be studied in conjunction with the Skeleton Core Map.

Each entry in the PNT is four words in length. Since the System Director is the first (and constitutes the minimum) entry in a PNT, let us examine this entry. From the dump we see

22A0 / 4259	=	Name (SYDIR) of System Director in truncated EBCDIC form
0024	=	Number of words in PNT (hexadecimal value)
0000	=	Not used

Consider another entry, SCAN2. The dump shows

220C / 1572	=	Name (SCAN2) of this core load in truncated EBCDIC form
0C2E	=	Core load word count
139C	=	Disk drive and sector address where core load resides

In turning to the ETV, we see that each entry is three words long. Consider the entry DISKN. From the dump we see

0000	=	Entry point to the subroutine ETV entry
4480	=	BSI indirect branch
00B8	=	Address where subroutine branches through

Figure 82 also illustrates the Core Load Data Words (CDW), the Core Load IBT, the Core Load ETV, and the FORTRAN I/O table.

Executive Branch Table (EBT). A map of all LIBF and call subroutines in the user subroutine and System Director areas of the skeleton. Each entry is one word long. The EBT is employed as a transfer vector: an indirect branch through the EBT is used to enter the referenced subroutine.

Skeleton Interrupt Branch Table (SKIBT). A map of all interrupt servicing subroutines in the skeleton. It is used in conjunction with the Master Branch Table (core load header words in sectors 1 and 2 on disk) to guide the interrupt to its proper routine in the skeleton. Each entry consists of a two-word BSC instruction.

The table is built during skeleton build time by entries put into the MBT. Word 2 of each entry is filled by a word from the corresponding MBT entry, while the location of the SKIBT entry itself replaces the MBT entry.

SKELETON CORE SIZE

The length of the skeleton in words is defined by the user at DEFINE CONFG time, and given by the parameter LSKEL (see System Design Considerations: Disk System Configuration). LSKEL must be an even value and is equal to the address of the first word in variable core (VCORE).

In general, skeleton size is estimated by the user after making allowances for the System Director, Skeleton I/O, Skeleton Common, user-written subroutines, and the PATCH area. Skeleton Common and the PATCH area dimensions are determined by exact user requirements. Some of the important factors influencing skeleton size will now be considered.

Core Storage

The amount of core storage available determines the number of features which can be included in a TSX system (see System Design Considerations: System Director). For example, is the system an off-line or an on-line system? Is time-sharing required? Is the Interval Timer Control feature used?

As a rule, in an on-line system using time-sharing, variable core must be a minimum of 3692 words. For example, in an 8K system, the maximum skeleton size would then be 4500 words (see Figure 83).

If the time-sharing feature is not required in an on-line system, the in-core skeleton may occupy all of core less 2500 words. This 2500-word restriction is necessary because of space requirements for the Cold Start and Error programs at the high end of core storage (see Figure 84). Under this system, non-process work would have to be accomplished off-line under TASK control (see Figure 85).

The maximum size of the skeleton is always dictated by the balance of core storage above the skeleton: 3692 words minimum for Nonprocess Monitor use, or for non time-sharing users, a minimum of 2500 words minimum for the Cold Start and Error programs.

It should be noted that too small a skeleton may demand frequent disk exchanges requiring excessive time, while too large a skeleton may reduce the variable space available for core loads and thus cause over-segmentation.

Figures 83, 84, and 85 summarize these rules for the on-line system (with time-sharing), on-line system (non-time-sharing) and the off-line system.

Inclusion of Subroutines in the Skeleton

The user may elect to include interrupt and other subroutines permanently in the skeleton for more rapid system response. The criteria governing these inclusions and the advantages gained thereby are discussed below.

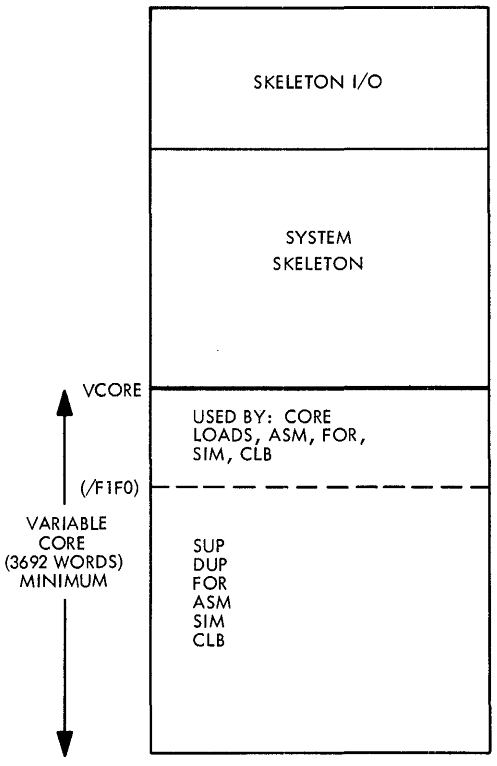

SKELETON I/O
SYSTEM SKELETON
VCORE
USED BY: CORE LOADS, ASM, FOR, SIM, CLB
(/F1F0)
VARIABLE CORE (3692 WORDS) MINIMUM
SUP
DUP
FOR
ASM
SIM
CLB

Figure 83. On-Line (Time-Sharing) System

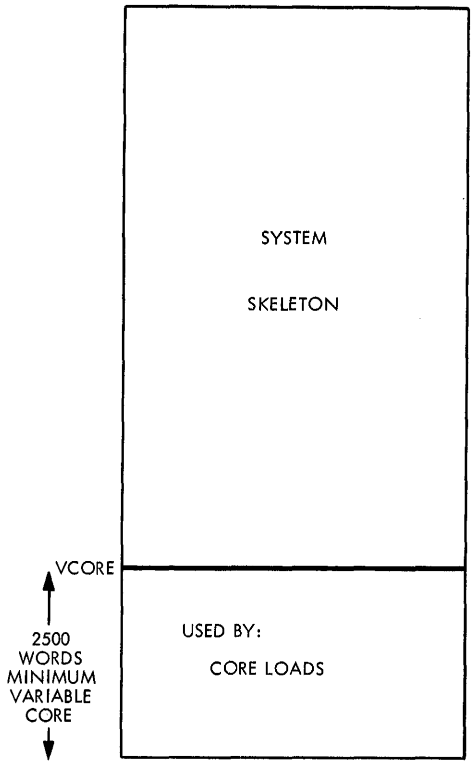


Figure 84. On-Line (Non Time-Sharing) System

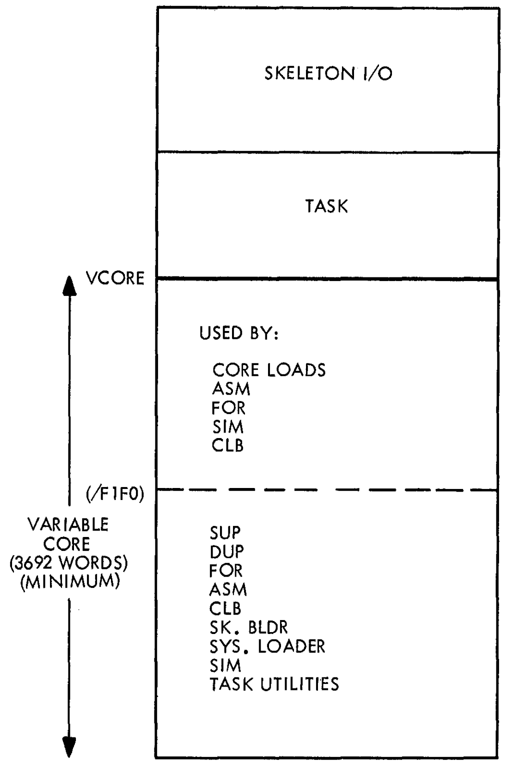


Figure 85. Off-Line System

Response Time. The shortest response time (that is, the minimum time before an interrupt servicing routine is entered after a process or program interrupt is recognized) is obtained by placing the servicing routine in core with the skeleton. The main advantage of having as many routines as possible permanently resident in the skeleton is faster response time.

Commonly-used Subroutines. Subroutines which are commonly enough referenced in core loads warrant inclusion in the skeleton. The advantages derived are:

1. Better utilization of disk user area and core load area. Every subroutine that is included in the skeleton is commonly shared by the several core loads referencing that subroutine. This means that each core load no longer contains the called subroutine, thus reducing core load length, and hence disk space.
2. Since the length of individual core loads has been shortened, less disk time is required to load the core load.
3. The effect of having a larger skeleton produces a disk space advantage. A larger Core Load Area is now possible because save areas, that is Special Save (SPSV), Process Save (PRSV) and Nonprocess Save (NPSV) can be smaller. Of necessity, the Interrupt Save (INSV) Area (as defined by the DEFINE CONFG parameter (LINS) can be no longer than the above-mentioned save areas. It can now be smaller. Furthermore, the user can now delete those subroutines that are included in the skeleton from the user area. This extra space on disk thus gained may be utilized for other purposes such as work areas and relocatable programs.

Subroutines Specified in the CALL TIMER Statement. Timer servicing subroutines should, as a rule, always be included in the skeleton. These subroutines can perform some execution, but it is preferred, if any I/O device is required, that they simply set a programmed interrupt (by a CALL LEVEL (I)) or queue a mainline core load and return control to ITC.

Consider for example where a mainline calls a timer as follows:

```
    EXTERNAL SUB1
    CALL TIMER (SUB1, 1, INCRE)
```

The user is responsible to ensure that the mainline that requested the CALL TIMER statement remain in core until the end of the elapsed specified time (that is, until the timer times out). One way of achieving this is for him to mask out all out-of-core interrupt levels and not change core loads until the timer interrupts. He thereby ensures that the core load containing the subroutine SUB1 remains in core.

If, however, he does not wish to remain in a masked state, a second approach is to have previously included SUB1 in the skeleton. In this way, he does not incur the penalty of waiting for the timer to interrupt, and also gains the advantage of not tying up variable core.

Count Subroutines. The Count subroutine is simply another method of servicing an event. Unlike timer interrupts, count interrupts may run off a different time base, and utilize larger time intervals. If the user plans to use the CALL COUNT statement, he should remember that for immediate servicing of an event, it is preferred that he set up these routines as part of his skeleton. If he does not, the event can be recorded and subsequently serviced by a CALL QIFON.

I/O Devices and their Associated Conversion Subroutines Required for FORTRAN DP I/O. On an on-line TSX system, the normal DP I/O utility functions are carried out by a package of skeleton I/O routines which also forms the basis of input-output operations in TASK. That is, the same set of I/O routines exist in TASK and the System Skeleton. Since the TSX System is a disk operating system, DISKN will be in core (that is, in the skeleton) at all times. If a 1053 printer or a 1816 keyboard/printer has been specified in the assignment stage, TYPEN/WRTYN must also be permanently in core. Similarly, if a 1443 printer has been assigned, PRNTN must be resident in the skeleton I/O area. These routines are automatically included in the System Skeleton at skeleton build time.

Although CARDN is always resident in core during TASK execution, this routine is an optional part of the Skeleton I/O, depending on user requirements. The user must, however, define at TASK assembly time (by equating CDINS to 1 or 0) whether or not CARDN is to be in the skeleton. If it is not included, it will be loaded as a part of those core loads which address the card reader. Note that in this event, the non-process components (such as the Assembler, FORTRAN Compiler, etc.) use their own card I/O routine.

If the user intends to do FORTRAN DP I/O from the skeleton, he should ensure that the conversion subroutine associated with any DP I/O device used by skeleton subroutines be included in the skeleton. He does this by means of an *INCLD control card at skeleton build time which loads the appointed conversion routine from the IBM Subroutine Library.

Figure 86 shows the relationship between each DP I/O device and its associated pair of function and conversion subroutines. For example, the 1442 card/read punch unit is associated with CARDN (its function routine) and HOLEB (its conversion routine). Note that there is no conversion routine for the 2401 magnetic tape drive: the conversion is by-passed. Conversion routines should be consistent with the precision required (that is, whether standard or extended) as specified in the TASK equate card PRICS.

Inclusion of Explicit and Implicit Subroutines. In the compilation of a FORTRAN problem program, the compiler-generated machine language coding includes a large number of branch instructions which transfer control to subroutines during execution of that program. It is, in fact, the subroutines that perform the majority of operations in any given problems. These subroutines can be classified into two distinct types: explicit, and implicit.

Explicit subroutines are those subroutines that are clearly formulated or externally visible in a main program. Implicit subroutines, on the other hand, are those subroutines which are involved in the solution of a problem program, but not externally revealed.

ASSOCIATED SUBROUTINES	DP I/O DEVICE
TYPEN EBPRT	1053/1816 Keyboard Printer
CARDN HOLEB	1442 Card/Read Punch
PRNTN EBPRT	1443 Printer
PAPTN PAPEB	1054/1055 Paper Tape Reader & Punch
MAGT	2401 Magnetic Tape Unit
TYPEN HOLEB	1816 Keyboard Unit
PLOTX ECHR1 or FCHR1	1627 Plotter Unit

Figure 86. Illustrating Relationship of DP I/O Devices to Associated Function and Conversion Subroutines

If the user plans to include FORTRAN subroutines in the Skeleton, he should make adequate core space allowance both for explicitly named and implicitly called subroutines. The explicitly named user-written subroutine is included in the Skeleton by specifying its name in an *INCLD control card at skeleton build time, while any implicitly referenced subroutines will be automatically loaded at the same time.

The following two examples examine skeleton core requirements for typical situations involving explicit and implicit subroutines.

EXAMPLE 1 - FORTRAN CASE. Consider a main program which is required:

1. To set up variables of a 10 by 10 matrix and
2. To call a user-written subroutine MSQRT which is to compute the square root of each element in the array.

Program Listing No. 5 shows the compilation run, from which it can be seen that if the user intends to include the subroutine MSQRT in the skeleton, he should make appropriate skeleton space allowances for the following:

(Explicit)	MSQRT	=	(program + variable) 72 words
(Implicit)	FSQRT	=	86 words
	FSTOX	=	102 words
	SUBSC	=	44 words
	SUBIN	=	36 words
			340 words

PROGRAM LISTING NO. 5: EXAMPLE 1 -- FORTRAN CASE

```
// JOB
// FOR MATRX
*NONPROCESS PROGRAM
*LIST ALL
C      SAMPLE MAIN PROGRAM TO CALL A MATRIX SQUARE ROOT SUBROUTINE
       DIMENSION VALUE(10,10)
       N = 10
       SUM = 0.0
       DO 5 I=1,N
       DO 5 J=1,N
       SUM = SUM + 1.0
     5 VALUE(I,J) = SUM
       CALL MSQRT(VALUE,N)
       CALL EXIT
       END
VARIABLE ALLOCATIONS
 VALUE(R )=00C6-0000   SUM(R )=00C8          N(I )=00CA          I(I )=00CC          J(I )=00CE

STATEMENT ALLOCATIONS
 5     =00F0

FEATURES SUPPORTED
 NONPROCESS

CALLED SUBPROGRAMS
 MSQRT   FADD    FLD     FSTO    FSTOX   SUBSC

REAL CONSTANTS
  .000000E 00=00D2    .100000E 01=00D4

INTEGER CONSTANTS
    10=00D6      1=00D7

CORE REQUIREMENTS FOR MATRX
 COMMON       0  INSKEL COMMON      0  VARIABLES     210  PROGRAM      66


 END OF COMPILATION


MATRX
DUP FUNCTION COMPLETED
// FOR MSQRT
*LIST ALL
*NONPROCESS PROGRAM
       SUBROUTINE MSQRT(A,N)
C      USER WRITTEN MATRIX SQUARE ROOT SUBROUTINE
       DIMENSION A(10,10)
       DO 1  I=1,N
       DO 1  J=1,N
     1 A(I,J) = SQRT(A(I,J))
       RETURN
       END
VARIABLE ALLOCATIONS
      I(I )=0000          J(I )=0002

STATEMENT ALLOCATIONS
 1     =001A

FEATURES SUPPORTED
 NONPROCESS

CALLED SUBPROGRAMS
 FSQRT   FSTOX   SUBSC   SUBIN

INTEGER CONSTANTS
      1=0006

CORE REQUIREMENTS FOR MSQRT
 COMMON       0  INSKEL COMMON      0  VARIABLES       6  PROGRAM      66


 END OF COMPILATION


MSQRT
DUP FUNCTION COMPLETED
*STORE              MSQRT
MSQRT
DUP FUNCTION COMPLETED
*STORECIL           MATRX MATRX
*CCEND
```

```
CLB, BUILD MATRX

CORE LOAD  MAP
TYPE NAME  ARG1   ARG2

*CDW TABLE 4002   000C
*IBT TABLE 400E   0023
*FIO TABLE 4031   0010
*ETV TABLE 4041   000C
*VTV TABLE 404D   0021
*PNT TABLE 406E   0004
MAIN MATRX 414C
PNT  MATRX 4070
LIBF FLD   41EA   404D
LIBF FSTO  41D0   4050
LIBF FADD  421E   4053
LIBF SUBSC 429C   4056
LIBF FSTOX 4186   4059
CALL MSQRT 42D0
LIBF FARC  4310   405C
LIBF SUBIN 4344   405F
CALL FSQRT 4386
LIBF FMPY  43D9   4062
LIBF FLDX  41E5   4065
LIBF FDIVX 4416   4068
LIBF FADDX 4218   406B
CALL FTNTR 4480
CALL FTRTN 449A
CORE       44AA   3B56

CLB, MATRX LD XQ

DUP FUNCTION COMPLETED
// XEQ MATRX   FX


// JOB
// END OF ALL JOBS
```

If other core loads are going to use the same subroutines, he may also include some of the implicit subroutines referenced by the main program MATRX. This will increase his skeleton size by 362 words as shown below.

(Implicit)	FADD	=	158 words
	FLD	=	102 words
	FSTO	=	102 words
			362 words

EXAMPLE 2 -- ASSEMBLER CASE. Consider a main program which is required:

To call a user-written subroutine QUAD to solve a quadratic equation

$$AX^2 + BX + C = 0$$

using the positive square root, and assuming the quantity under the square root sign is greater than zero.

From Program Listing No. 6, it can be seen that if the user intends to include QUAD in the skeleton, adequate space allowances should be made for the following:

(Explicit)	QUAD	=	(program + variables) 82 words
	FLD	=	102 words
	FMPY	=	65 words
	FSTO	=	102 words
	FSUB	=	158 words
	FSQR	=	86 words
	FADD	=	158 words
	FDIV	=	106 words
(Implicit)	FARC	=	52 words
	FSTOX	=	102 words
	FLDX	=	102 words
	FDIVX	=	106 words
	FADDX	=	158 words
	FTNTR	=	40 words
	FTRTN	=	40 words
			1459 words

PROGRAM LISTING NO. 6: EXAMPLE 2 -- ASSEMBLER CASE

```
// JOB
// ASM
  *LIST
                     *              SAMPLE SUBROUTINE TO SHOW THE USE OF
                     *              IMPLICIT AND EXPLICIT SUBROUTINE
                     *              CALLS
                     *
                     *              THIS SUBROUTINE SOLVES A QUADRATIC
                     *              EQUATION
                     *                  A*(X**2) + B*X + C = 0
                     *              USING THE POSITIVE SQUARE ROOT, AND
                     *              ASSUMING THE QUANTITY UNDER THE
                     *              SQUARE ROOT SIGN IS GREATER THAN ZERO
                     *
  0012     18901100          ENT      QUAD
  0000     0000              BSS   E  0
  0000 00 00000000   A       DEC      0.0         ARG A WILL BE STORED HERE
  0002 00 00000000   B       DEC      0.0         ARG B WILL BE STORED HERE
  0004 00 00000000   C       DEC      0.0         ARG C WILL BE STORED HERE
  0006 00 00000000   X       DEC      0.0         RESULT WILL BE STORED HERE
  0008 00 00000000   TEMP1   DEC      0.0
  000A 00 00000000   TEMP2   DEC      0.0
  000C 00 40000083   FOUR    DEC      4.0
  000E 00 40000082   TWO     DEC      2.0
  0010     0002      TEMP    BSS      2
                     *                             SET UP ARGUMENTS A,B,C
  0012 0  0000       QUAD    DC       0
  0013 01 C4800012           LD    I  QUAD         VALUES CCME FROM MAIN PROG
  0015 0  D8FA               STD      TEMP
  0016 01 CC800010           LDD   I  TEMP
  0018 0  D8E7               STD      A
  0019 01 74010012           MDX   L  QUAD,1
  001B 01 C4800012           LD    I  QUAD
  001D 0  D8F2               STD      TEMP
  001E 01 CC800010           LDD   I  TEMP
  0020 0  D8E1               STD      B
  0021 01 74010012           MDX   L  QUAD,1
  0023 01 C4800012           LD    I  QUAD
  0025 0  D8EA               STD      TEMP
  0026 01 CC800010           LDD   I  TEMP
  0028 0  D8DB               STD      C
                     *                             TO CALCULATE DIVISOR 2*A
  0029 20 064C4000           LIBF     FLD          LOAD 2.0 INTO FAC
  002A 1  000E               DC       TWO
  002B 20 06517A00           LIBF     FMPY         MULTIPLY 2.0 BY A TO GET
  002C 1  0000               DC       A            2.0*A
  002D 20 068A3580           LIBF     FSTO         STORE DIVISOR TEMPORARILY
  002E 1  0008               DC       TEMP1
                     *                             TO CALCULATE DIVIDEND = A +
                     *                             SQUARE ROOT OF B**2 -4*A*C
  002F 20 064C4000           LIBF     FLD          LOAD A INTO FAC TO CALCU-
  0030 1  0000               DC       A            LATE 4*A*C
  0031 20 06517A00           LIBF     FMPY
  0032 1  0004               DC       C
  0033 20 06517A00           LIBF     FMPY
  0034 1  000C               DC       FOUR
  0035 20 068A3580           LIBF     FSTO         STORE 4*A*C TEMPORARILY
  0036 1  000A               DC       TEMP2
  0037 20 064C4000           LIBF     FLD          LOAD B INTO FAC TO CALCU-
  0038 1  0002               DC       B            LATE B**2 = B*B
  0039 20 06517A00           LIBF     FMPY
  003A 1  0002               DC       B            HAVE B*B IN FAC. NOW TO
  003B 20 068A4080           LIBF     FSUB         SUBTRACT 4*A*C
  003C 1  000A               DC       TEMP2
  003D 30 06898640           CALL     FSQR         TAKE SQ.RT.OF B*B-4*A*C
  003F 20 06044100           LIBF     FADD         ADD A TO SQUARE ROOT. GET
  0040 1  0000               DC       A            A+SQ.RT(B*B - 4*A*C)
                     *                             TO DIVIDE DIVIDEND BY 2*A
  0041 20 06109940           LIBF     FDIV
  0042 1  0008               DC       TEMP1
                     *                             TO STORE RESULT IN X
  0043 20 068A3580           LIBF     FSTO
  0044 1  0006               DC       X
  0045 01 74010012           MDX   L  QUAD,1
  0047 01 C4800012           LD    I  QUAD
  0049 0  D0C6               STO      TEMP
  004A 0  C8BB               LDD      X
  004B 01 DC800010           STD   I  TEMP
  004D 01 74010012           MDX   L  QUAD,1
  004F 01 4C800012           BSC   I  QUAD
  0052                       END

      NO ERRORS IN ABOVE ASSEMBLY.
QUAD
DUP FUNCTION COMPLETED
// ASM MAIN
  *LIST
```

```
                       *            MAIN PROGRAM TO CALL SUBROUTINE QUAD
  0000       0000                BSS   E  0
  0000 00 40000081     AA        DEC      1.0
  0002 00 60000082     BB        DEC      3.0
  0004 00 40000082     CC        DEC      2.0
  0006 00 00000000     XX        DEC      0.0         TO BE FILLED IN BY SUBR
  0008 30 18901100     START CALL         QUAD
  000A 1  0000                   DC       AA
  000B 1  0002                   DC       BB
  000C 1  0004                   DC       CC
  000D 1  0006                   DC       XX
  000E 0  3000                   WAIT
  000F 30 059C98C0               EXIT
  0012    0008                   END      START

     NO ERRORS IN ABOVE ASSEMBLY.
MAIN
DUP FUNCTION COMPLETED
// XEQ MAIN  L
*CCEND

 CLB, BUILD MAIN

 CORE LOAD  MAP
 TYPE NAME  ARG1   ARG2

 *CDW TABLE 4002   000C
 *IBT TABLE 400E   0023
 *FIO TABLE 4031   0010
 *ETV TABLE 4041   000C
 *VTV TABLE 404D   0021
 *PNT TABLE 406E   0004
 MAIN MAIN  407A
 PNT  MAIN  4070
 CALL QUAD  4096
 LIBF FLD   413A   404D
 LIBF FMPY  4153   4050
 LIBF FSTO  4120   4053
 LIBF FSUB  41A4   4056
 CALL FSQR  4246
 LIBF FADD  41B0   4059
 LIBF FDIV  4289   405C
 LIBF FARC  42EE   405F
 LIBF FSTOX 40D6   4062
 LIBF FLDX  4135   4065
 LIBF FDIVX 4284   4068
 LIBF FADDX 41AA   406B
 CALL FTNTR 4322
 CALL FTRTN 433C
 CORE       434C   3CB4

 CLB, MAIN  LD XQ


// JOB
// END OF ALL JOBS
```

CALCULATING SKELETON CORE SIZE

Skeleton Core Size can be arrived at by computing the value of the start address of variable core (VCORE), an even number. VCORE is equal to a summation of:

```
SKELETON  +   INSKEL  +   SYSTEM   +   USER'S   +  SKELETON  +  PATCH
  I/O         COMMON     DIRECTOR     PROGRAMS      TABLES       AREA
```

The manner in which the SYSTEM DIRECTOR is calculated has already been discussed (see System Design Considerations: System Director). INSKEL COMMON, the PATCH AREA, and the USER'S PROGRAMS are defined by the user.

```
SKELETON I/O = Fixed Area = 200 words
               +Disk Device Tables = 66 words per drive
               +DISKN = 900 + (12 + (9XDORG2)) X DORG1 + 12 X MKLEV
               + 1053 Device Tables = 14 + message unit size (add 1 if the message unit size is odd)
                                           per 1053 and 1816
               + 1816 Device Tables = 14 words per keyboard
               + TYPEN = 256 + (3XN) + (341 + (8XNOCYL)) X NOBUF + (20 + (2XTORG))
                         X(1-NOBUF) + (418 + (8X(M-1))) X TORG + 16XMKLEV
                         where N = number of 1053/1816s
                         and M = number of 1816s
               + 1053/1443 Timing Response Routine = 2 + (78XTORGN) + 22XPORG
               + 1443 Device Table = 14 words
               + PRNTN = 323 words + 16XMKLEV
               + Constants & Work Areas = 200 + 42XECPT1 + 27 (1-ECPT1)
                                          where ECPT1 = 0 or 1 if EAC printer
                                          is a 1053 or 1443 respectively
               + CARDN (if included) = 328 words + 23XCRDNO + 10XMKLEV
```

Note that NOBUF, TORG, TORGN, MKLEV, PORG, NOCYL, and ECPT1 are TASK equate cards.

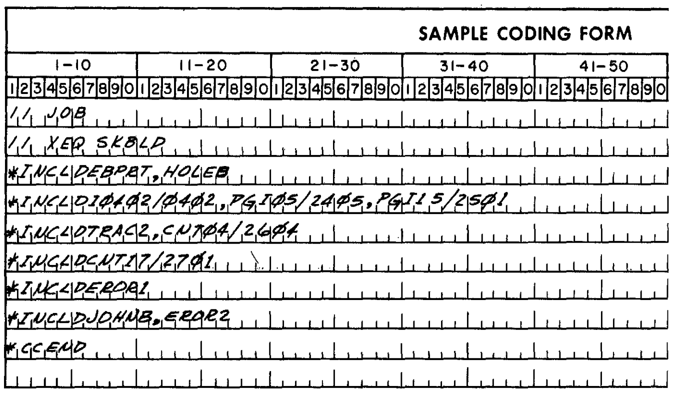
```
SKELETON TABLES = Program Name Table = 4 + 4 X the number of called mainline and combination
                                       core loads from the skeleton.
                  + Executive Transfer Vector = 3 X the number of LIBF functions in the skeleton.
                  + Executive Branch Table = 1 word per entry for each entry point for LIBF and
                                             CALL Subroutines contained in the skeleton (excepting
                                             Skeleton I/O).
                  + Skeleton Interrupt Branch Table = 2 X the number of bits in all labels (that is,
                                                      ILSW words) NB00-23.
```

USE OF *INCLD CONTROL CARDS

Before the System Skeleton can be built, it is a condition that user-written subroutines and programs intended for inclusion in the skeleton must be assembled and/or compiled and stored on disk in relocatable format. These are assigned to the skeleton area by *INCLD control cards at skeleton build time. The various types of subroutines that are suitable candidates for inclusion have already been mentioned elsewhere in this section.

Subroutines planned for tracing and/or error options may also be included in the skeleton, but it should be noted that they are not, as such, automatically functional, because these routines can only be entered through linkages provided in the individual core loads. In order to use them, therefore, their names must appear in appropriate *INCLD cards which must be present when building those core loads where tracing or error options are desired. Furthermore, in the case of error subroutines, the entry is made only if the core load is a process mainline, combination or interrupt core load. If any nonprocess core load or monitor function is in progress, the error routine will not be included.

The following example is given to illustrate the general use of *INCLD control cards in a typical skeleton build operation. The sequence of control cards is shown below:


SAMPLE CODING FORM

```
1-10      11-20     21-30     31-40     41-50
1234567890123456789012345678901234567890123456789 0
// JOB
// XEQ SKBLD
*INCLDEBPRT,HOLEB
*INCLDIØ4Ø2/Ø4Ø2,PGIØ5/24Ø5,PGI15/25Ø1
*INCLDTRAC2,CNTØ4/26Ø4
*INCLDCNT17/27Ø1
*INCLDEROR1
*INCLDJOHNB,EROR2
*CCEND
```

Five types of subroutines are considered:

- EBPRT and HOLEB are IBM Library Conversion Subroutines. JOHNB is a special user-written arithmetic subroutine.
- I0402/0402 serves as an interrupt servicing subroutine for a process interrupt assigned to level 04, bit 02.
- PGI05/2405 and PGI15/2501 are interrupt servicing routines which service programmed interrupts on levels 5 and 15 respectively.
- CNT04/2604 and CNT17/2701 are user-written count routines #4 and #17 respectively.
- TRAC2, EROR1 and EROR2 are special user-written trace and error subroutines. Note that at core load build time, these three subroutines must be named by *INCLD cards in those core loads selected for the trace and/or error options in order to establish their linkages in the skeleton. (If this is not done, no tracing or error checking takes place.)

  The flexibility in system usage and design (in permitting a core load to be traced by a special user-written trace routine other than the trace routine normally included in the skeleton) is a definite advantage as it is simpler to modify a core load than to modify the skeleton.

SUMMARY OF THE SKELETON BUILD PROCESS

Before the System Skeleton can be built, several prerequisite conditions must have been met. These include:

- The IBM Nonprocess System has been loaded.
- User-written subroutines required for residence in the skeleton have been assembled/compiled and stored on disk in relocatable format.
- The System Director has been assembled and stored on disk in relocatable format.
- Disk file configuration has been defined.
- An Operating TASK (that is a user-configurated TASK) has been assembled and punched into cards.

Briefly, the skeleton build function can be broken down (in simplified form) into three separate steps:

1. TASK Initialization
2. Relocation of component parts
3. Building the Skeleton Tables

Figure 87 illustrates the skeleton build operation as each sector of the skeleton is constructed and relocated to the Skeleton Area on disk.

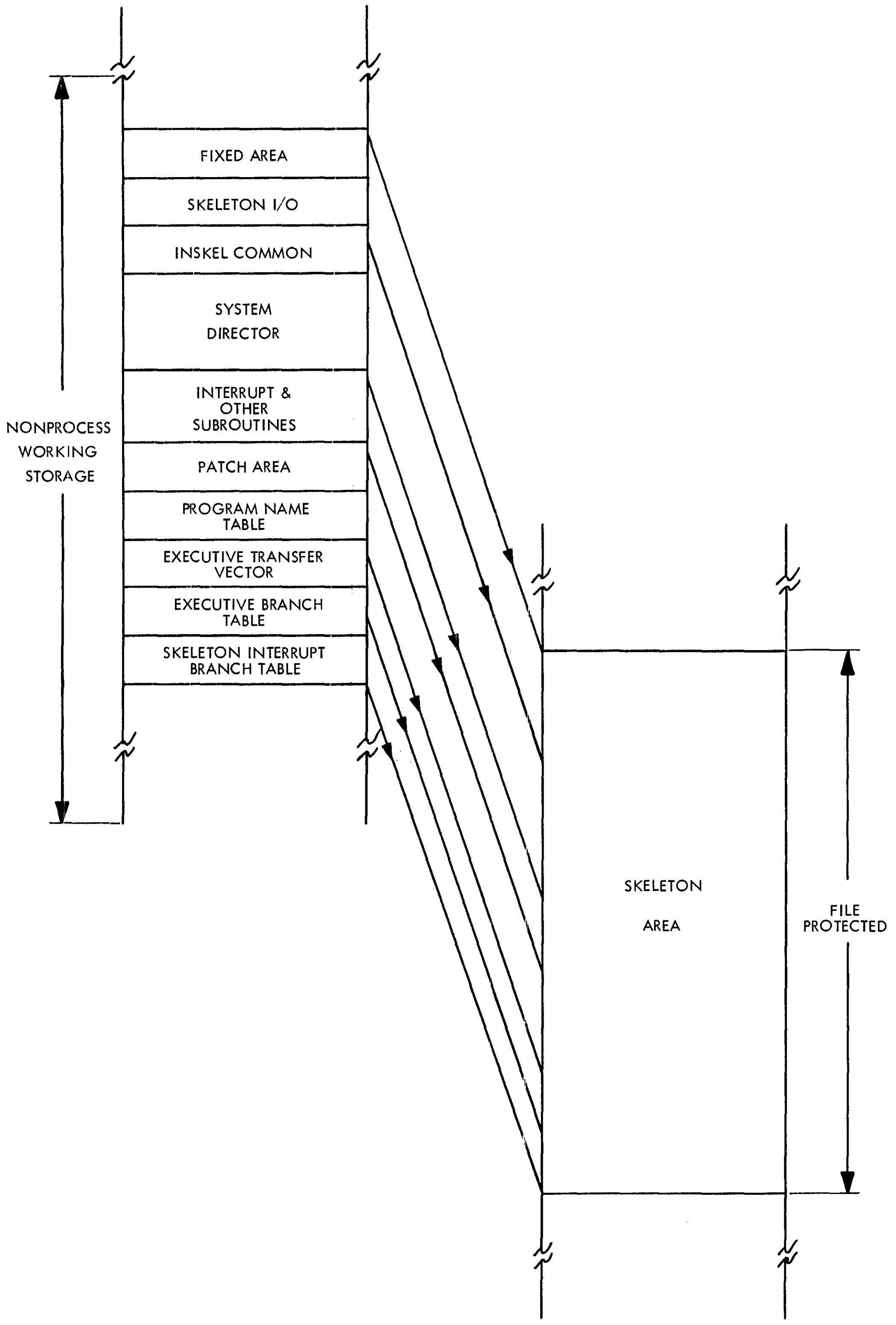


Figure 87. Layout of the System Skeleton as it would appear at Skeleton Build Time in NPWS and the Skeleton Area

TASK Initialization. TASK begins the skeleton build function by initializing certain communications areas which reside in sectors 1 - 4 on disk. These areas include:

- Master Communications Words (MCW)
- Master Branch Table (MBT)
- Skeleton FORTRAN I/O Table (SFIO)
- Executive Transfer Vector (ETV)
- Skeleton Subroutine Name Table (SKSUB)

This step determines the amount of core storage available between the high addressed end of TASK and the first word of the Skeleton Builder: the available core storage is cleared and subsequently allocated to the various loading tables. The Skeleton Subroutine Table (SKSUB) is then read from disk into its allocated area in core. The Control Record Entry Table (CRENT), which was constructed by the Skeleton Build Supervisor phase (SKA), is read, one entry at a time. These are analyzed and inserted into the appropriate tables.

Relocation of TSX Component Parts. The main function of the skeleton build operation is the construction in core-image format in the Nonprocess Working Storage on disk of the permanent part of all core loads and to establish their linkages to the System Director and various other IBM and user-written programs. As each sector of the skeleton is built it is written to the Nonprocess Working Storage. The disk image of the completed skeleton is finally physically moved to the Skeleton Area of the process portion of disk and file protected.

The Skeleton I/O is first written to a predefined area in the Nonprocess Working Storage on the temporary disk drive for later reference by the Skeleton Builder.

The length and initial address of Skeleton Common are now obtained from the Master Communications Area, and the desired size of the Common Area in the skeleton image cleared and reserved. The upper boundary of this area is later set to correspond with the starting address of the System Director.

It was noted in System Design Considerations: System Director that the System Director was assembled and stored on disk prior to a skeleton build. At this same time, space was allocated for an Interrupt Core Load Table (ICLT), its size being determined by user-specified NIL equate cards. Using the System Director as the main program, the Skeleton Builder constructs the skeleton in much the same fashion that the Core Load Builder builds a core load. Certain words in the ICL table (the first header word and the first and second entry words) associated with bits corresponding to those interrupt subroutines resident in the skeleton are now filled in. Later on, DUP makes the required entries into the ICL table whenever it stores an interrupt core load on disk.

User-written and other subroutines are now included in the skeleton image on disk following the System Director.

The Patch area constitutes that portion of core image that remains between the end of the subroutine area and the Skeleton Program Name Table.

Building the Skeleton Tables. During the final stage of the skeleton build process, the four tables shown in Figure 81 are modified and/or built from data provided in the Master Communications Area and user-specified *INCLD control cards. Note that the tables are built and used by the Skeleton Builder during the construction of the skeleton core load. All of the loading tables are assembled in a descending chain; that is, the first entry occupies higher core locations than the second, the second higher than the third, and so on. The details and function of each table are given elsewhere in this section (see Constitution of the System Skeleton).

For detailed step-by-step operational procedures of the skeleton build process, the user should refer to the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Rebuilding the System Skeleton

While relocatable programs can be deleted and replaced on-line by the Nonprocess Monitor, it is not possible to modify any features of the System Skeleton on-line. Changes in the skeleton area (including skeleton interrupt routines) will thus require an off-line Skeleton rebuild.

The TSX Skeleton may be rebuilt at any time by following the detailed procedures specified for an initial Skeleton build in the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Since INSKEL COMMON is not open-ended, the user may face the difficulty of adding to it once it is defined. It is recommended that an extra area should be reserved in INSKEL COMMON to allow for programming contingencies. See Core Load Rebuild Conditions.

The Executive Branch Table (EBT) and the Skeleton Interrupt Branch Table (SKIBT) have already been described. These tables are provided to allow the user to rebuild the System Skeleton if he were modifying subroutines, changing the logic flow or adding patches to the System Director and TASK such that addresses in core loads will still reference a fixed address in variable core. An ability to shift the entry points of subroutines within the Skeleton is thus available without the necessity of rebuilding the referencing core loads.

If, however, the entry points within these tables no longer pointed to the same subroutine, all core loads must be rebuilt.

Core Load Rebuild Conditions

When the Skeleton is initially built, the entry points to the in-core-with-skeleton interrupt routines are placed in the ICL Table, and all other table entries are cleared.

When this skeleton is later rebuilt, the in-core-with-skeleton entry points are substituted into the proper areas; the word count and sector address of out-of-core interrupt routines are then recovered from the old skeleton and placed in their respective table locations, provided that there is not already an entry point there.

In a skeleton rebuild, it may not be necessary to rebuild those core loads built under the previous (old) Skeleton if the following conditions are met:

1. No previously included CALL or LIBF type subroutines may be removed from the Skeleton. An in-core interrupt (ICI) or a count subroutine may be substituted, provided it has the same number of entry points and occupies the same relative position within the Skeleton as the deleted subroutine. Any core loads referencing the deleted subroutine must be rebuilt.
2. No additional LIBF or ISS subroutine (disk format types 3, 5, and 6) may be included in the new Skeleton. In-core interrupt (ICI) and CALL (type 4) subroutines may be added provided the patch area is large enough to contain the additions.
3. The number of entry points for included subroutines may not be modified between inclusions.
4. If TASK or the System Director is reassembled, the number of interrupt levels used, the length of INSKEL COMMON, and the length of the Skeleton should not be altered.

   Note that a change in the size of INSKEL COMMON implies a reassembly of TASK. Also, if the location of INSKEL COMMON changes, all core loads using INSKEL COMMON must be rebuilt. If, for example, Skeleton I/O changed in size, an adjustment in INSKEL COMMON size equal to the change in the Skeleton I/O could eliminate the necessity for rebuilding all core loads that use INSKEL COMMON.
5. If there is to be any variation in the number or order of *INCLD control cards, new *INCLD control cards must be prepared using the previous core map. They must specify (in order of occurrence in the previous map) all ICI, CALL, and LIBF names beginning with the first map entry following

   PNT SYDIR

   An alternative is to use the original *INCLD control cards in their original order. Additional ICI and CALL subroutines may be specified on following *INCLD control cards.
6. Following the rebuild process, a comparison of the previous Skeleton and new Skeleton core maps must show identical entry points (that is, ARG2 -- see Figure 88) for those LIBF and CALL map entries common to both skeletons.

Example of Initial Skeleton Build and Skeleton Rebuild

Figure 88 illustrates the general sequence of control cards, the Skeleton Core map, and the Interrupt Core Load (ICLT) map for a typical skeleton build/rebuild situation. A separate interpretation of these maps is given at the end of this section.

In the initial skeleton:

- INT01/0000 is an interrupt servicing routine for a process interrupt assigned to level 00, bit 00.

- INT02/2405 is an interrupt servicing routine which services a programmed interrupt on level 05.
- INT03/2500 services a programmed interrupt on level 14.
- SUB01 is a special user-written arithmetic subroutine.
- INT04/2600 is a user-written count routine #00.
- MASK and EBPRT are IBM Library Subroutines.

The four K13 level 1 Skeleton Builder error messages, following the ICL Table Map, are warnings to the user that core load names PML01, PML02, CCL01, and CCL02 referenced by calls in routines (e.g., CHAIN, SPECL, QUEUE) contained in the initial skeleton have not yet been built, and, therefore, are not entered in FLET (see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754).

In the rebuilt skeleton:

- No major modifications are implemented. Neither the Skeleton I/O nor INSKEL COMMON are altered.
- Four implicit routines (and those explicit routines referenced by these routines) are added. These are:
  1. SUB02 -- a user-written arithmetic routine.
  2. INT05/2701 -- a user-written count routine #17.
  3. DPART and UNMK are IBM Library Subroutines.
  4. These additions constitute entries to the ICL Table.

Note that SUB02 and INT05/2701 cannot reference or call any LIBF function subroutines as this violates the conditions stated (see Core Load Rebuild Conditions).

The user should be aware that in rebuilding a skeleton, the control cards must be in exact order-correspondence with the previous skeleton so that routines will be loaded in the same order-sequence and entry points in the transfer vector remain valid.

Interpretation of the Skeleton Core Map and the Interrupt Core Load Table (ICLT) Map

Skeleton Core Map

The Skeleton Builder always prints a map of the assembled skeleton formatted as follows:

```
SKEL   CORE   MAP          (Page Heading)
TYPE   NAME   ARG1   ARG2  (Column Heading)
```

Type indicates the map entry type (e.g., LIBF, CALL, PNT). Up to five alphameric characters are allowed under NAME to describe a subroutine, control program, etc. (e.g., DISKN, SYDIR).

Control Records For Initial Skeleton Build

```
//JOB 1       X
//XEQ SKBLD
*INCLDINT01/0000,MASK,EBPRT,INT02/2405
*INCLDSUB01,INT03/2500,INT04/2600.
*CCEND
```

Skeleton Core Map

```
SKEL    CORE     MAP
TYPE    NAME     ARG1     ARG2

LIBF    DISKN    021B     3EBB
LIBF    TYPEN    0582     3EBE
LIBF    WRTYN    0582     3EBE
LIBF    PRNTN    0A07     3EC1
LIBF    CARDN    0C71     3EC4
CALL    EXIT     1C9F     00B6
CALL    LINK     1CA1     008E
INSK             0DD4     10C1
PNT     SYDIR    10C2     3E26
ICI     INT01    1F80
ICI     INT02    1F91     1005
ICI     INT03    1F9E     1100
ICI     INT04    1FAF     1200
CALL    MASK     1FBA     3E44
LIBF    EBPRT    1FDC     3EC7
CALL    SUB01    207D     3E43
CALL    OUTTR    19DA     3E42
CALL    CHAIN    1BA7     3E41
CALL    INTEX    1AE1     3E40
CALL    SHARE    1C4F     3E3F
CALL    SPECL    1BEC     3E3E
CALL    BACK     1C03     3E3D
CALL    EACLK    1ED4     3E3C
CALL    QUEUE    2084     3E3B
PNT     PML01             3E2A
CALL    UNQ      2146     3E3A
PNT     PML02             3E2E
PNT     CCL01             3E32
PNT     CCL02             3E36
CALL    PRT      218C     3E39
PTCH             21D6     3E23


ICL TABLE MAP
LLBB    WC/EP    SA       ICLT

0000    1F80              11CC
1005    1F91              1296
1100    1F9E              12AC
1200    1FAF              12B4

 K13  PML01 LEV.1

 K13  PML02 LEV.1

 K13  CCL01 LEV.1

 K13  CCL02 LEV.1

 SK6, SYDIR LD NX
```

Control Records For Skeleton Rebuild

```
//JOB 1       X
//XEQ SKBLD
*INCLDINT01/0000,MASK,EBPRT, INT02/2405
*INCLDSUB01,INT03/2500,INT04/2600
*INCLDSUB02,DPART,INT05/2701,UNMK
*CCEND
```

Rebuilt Skeleton Core Map

```
SKEL    CORE     MAP
TYPE    NAME     ARG1     ARG2

LIBF    DISKN    021B     3EBB
LIBF    TYPEN    0582     3EBE
LIBF    WRTYN    0582     3EBE
LIBF    PRNTN    0A07     3EC1
LIBF    CARDN    0C71     3EC4
CALL    EXIT     1C9F     00B6
CALL    LINK     1CA1     008E
INSK             0DD4     10C1
PNT     SYDIR    10C2     3E22
ICI     INT01    1F80
ICI     INT02    1F91     1005
ICI     INT03    1F9E     1100
ICI     INT04    1FAF     1200
ICI     INT05    1FBD     1301
CALL    MASK     1FCE     3E44
LIBF    EBPRT    1FF0     3EC7
CALL    SUB01    2091     3E43
CALL    OUTTR    19DA     3E42
CALL    CHAIN    1BA7     3E41
CALL    INTEX    1AE1     3E40
CALL    SHARE    1C4F     3E3F
CALL    SPECL    1BEC     3E3E
CALL    BACK     1C03     3E3D
CALL    EACLK    1ED4     3E3C
CALL    QUEUE    2098     3E3B
CALL    UNQ      215A     3E3A
CALL    PRT      21A0     3E39
CALL    SUB02    21EB     3E38
CALL    DPART    21F4     3E37
CALL    UNMK     2202     3E36
PNT     PML01             3E26
PNT     PML02             3E2A
PNT     CCL01             3E2E
PNT     CCL02             3E32
CALL    QIFON    224C     3E35
CALL    VIAQ     22E6     3E34
PTCH             2345     3E1F

ICL TABLE MAP
LLBB    WC/EP    SA       ICLT

0000    1F80              11CC
0501    017A     04C6     11FA
0A02    017A     04C8     1230
1005    1F91              1296
1100    1F9E              12AC
1200    1FAF              12B4
1301    1FBD              12DA

 SK8, SYDIR LD XQ
```

Figure 88. Core Map for Initial and Rebuilt Skeleton

ARG1 and ARG2 may contain either a four-digit hexadecimal number or a blank field.

Program Name Table (PNT).

```
PNT     NNNNN    XXXX    YYYY
```

The word count and disk address of the core load named NNNNN which is referenced within the skeleton are assigned to locations YYYY and YYYY+1 of the skeleton PN Table. The first PNT entry is always the System Director where NNNNN = SYDIR and XXXX is the initial core location of the ICL Table pointer block which is identified in the System Director listing by the symbolic name -- COMA. The XXXX field is blank for all remaining PNT entries.

In-core-with-Skeleton Interrupt (ICI).

```
ICI    NNNNN    XXXX    LLBB
```

The entry point to the in-core-with-Skeleton interrupt servicing routine named NNNNN is at absolute location XXXX. LLBB designate the interrupt level and bit position within the ILSW for that associated level.

If LL is less than the number of interrupt levels (K) defined for the system, NNNNN is assigned to service the process interrupt on PISW bit position BB of level LL.

If LL = K or LL = K+1, NNNNN is assigned to service a programmed interrupt on level BB or BB+14 respectively. Note that BB is now used as a level designation.

If LL = K+2 or LL = K+3, NNNNN has been designated as count subroutine BB or BB+16 respectively. Note that BB is now used as a count subroutine number in the range 0-31.

For an ICI assigned to level 0 on PISW bit position 0, ARG2 will be printed as a blank field.

Library Function Subroutines (LIBF).

```
LIBF    NNNNN    XXXX    YYYY
```

The LIBF-type subroutine entry point named NNNNN is at absolute location XXXX of the skeleton. The corresponding three-word transfer vector entry point will be at location YYYY in variable core.

CALL-type Subroutines (CALL).

```
CALL    NNNNN    XXXX    YYYY
```

The CALL-type subroutine entry point named NNNNN is at absolute location XXXX of the skeleton. The indirect entry point is at location YYYY of the Skeleton Executive Branch Table (EBT).

INSKEL COMMON (INSK).

```
INSK             XXXX    YYYY
```

The low core storage boundary of INSKEL COMMON is at absolute location XXXX of the skeleton. The high boundary is at location YYYY.

Patch Area (PTCH).

```
PTCH             XXXX    YYYY
```

The patch area (that is, unused core locations) extends from the absolute location, XXXX, of the skeleton through location YYYY.

COMMON (COMM).

```
COMM             XXXX    YYYY
```

If any included subroutines contain references to COMMON, allocation is made between absolute locations XXXX and YYYY, using the standard method. It should be noted that these locations are in variable core; allowances must therefore be made in all core loads for overlapping results.

Interrupt Core Load Table (ICLT) Map

The ICL Table map is printed to reveal any interrupt assignments made in the skeleton ICLT. Its format is as follows:

```
ICL     TABLE    MAP
LLBB             WC/EP   SA   ICLT   (Column
                                      Heading)
```

The interrupt level and bit assignment are indicated by a four-digit hexadecimal number under LLBB. The two high-order digits contain the level; the two low-order digits represent the bit assignment.

If the entry is an in-core-with-Skeleton routine, the WC/EP column will contain the hexadecimal entry point to this routine. The SA field will be blank. The ICL Table absolute core location in which the entry point is placed is indicated in the ICLT column.

When rebuilding the skeleton with the SAVE ICL TABLE option, word counts and sector addresses of any interrupt core loads are retained from the old ICLT. Their interrupt assignments are indicated in the LLBB column. The WC/EP and SA columns will contain their word counts and disk addresses. The corresponding ICLT absolute core location is in the ICLT column.

To broaden the scope of this manual, and to facilitate its use by individuals of divergent backgrounds and experience, selected material emphasizing recommended practice and technique in the implementation of the IBM 1800 Time Sharing Executive System are presented in this section. This material directly supplements the concepts discussed so far in the text.

A separate chapter (Basic Concepts of Data Acquisition and Process Control Systems) is included for a two-fold purpose: (1) as an introduction to the TSX Sample System, and (2) to acquaint the inexperienced reader with the field of data acquisition and process control systems. It is not intended as an exhaustive study, and the reader is referred to further sources of information on this vast subject. The more experienced reader may prefer to scan this portion of the section as refresher material, or to skip it entirely.

The final chapter, TSX Sample System, is a comprehensive step-by-step example of a working TSX on-line system which touches on every aspect of TSX system concepts, design, and usage.

WRITING ASSEMBLER LANGUAGE SUBROUTINES

This chapter provides guidance to the user in the assembly and specification of user-written subroutines included either as additions or modifications to the TSX system. User-written assembler language subroutines must follow the writing specifications outlined below.

The subroutine source statements shown in the following examples should be preceded and followed by the following control cards for the assembly process.

```
// JOB
// ASM
*LIST
*PRINT SYMBOL TABLE
     .
     :
Subroutine Source Deck
     .
     :
// DUP
*STORE          NAME
```

Call Subroutines

A subroutine that is called by a CALL statement is linked to via a long BSI instruction. For example, a FORTRAN source statement

```
CALL SUB (I, J, K, 101)
```

or an Assembler language calling sequence

```
CALL              SUB
DC                ADDRI
DC                ADDRJ
DC                ADDRK
DC                ADCON
```

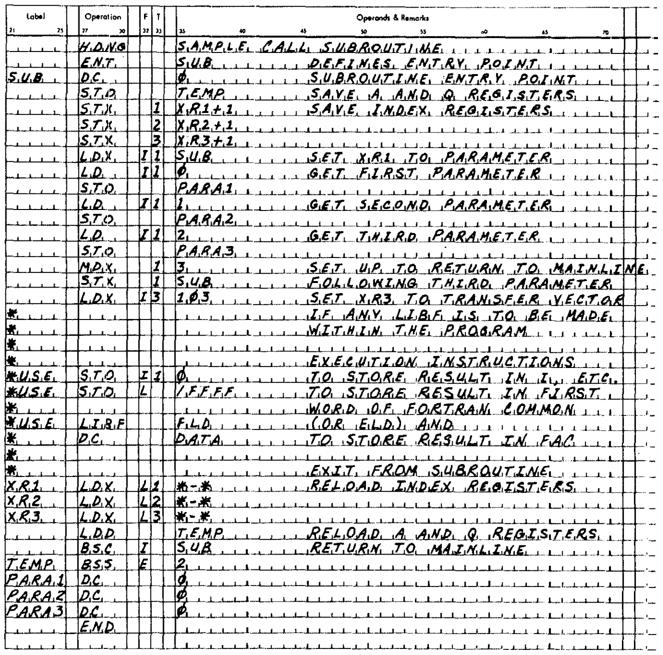
appears in core at execution time as

```
BSI    L          SUB
DC                ADDRI
DC                ADDRJ
DC                ADDRK
DC                ADCON
```

where ADDRI, ADDRJ, and ADDRK are the core addresses at which the variables I, J, and K are stored, and ADCON is the core address where the constant, 101, is stored.

Note that most subroutines entered by an Assembler language calling sequence expect the constants themselves to appear in the calling sequence rather than the address of the constants. Therefore, not all subroutines entered by a CALL can be called from a FORTRAN program.

The following example illustrates how to define the entry point, save the contents of the registers, get the parameters, and return to the calling program.



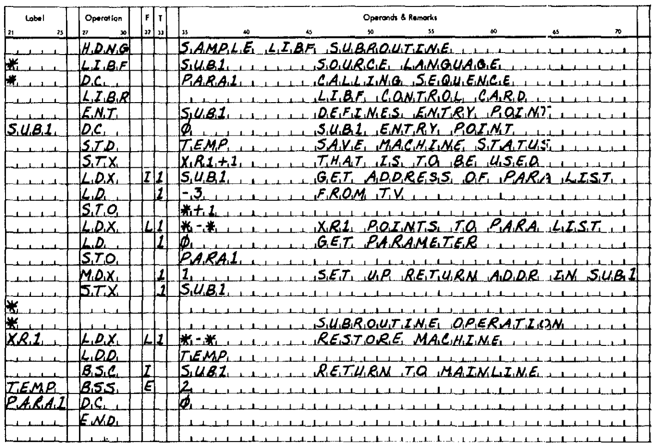
```
      HDNG     SAMPLE CALL SUBROUTINE
      ENT      SUB         DEFINES ENTRY POINT
SUB   DC       Ø           SUBROUTINE ENTRY POINT
      STO      TEMP        SAVE A AND Q REGISTERS
      STX   1  XR1+1       SAVE INDEX REGISTERS
      STX   2  XR2+1
      STX   3  XR3+1
      LDX  I1  SUB         SET XR1 TO PARAMETER
      LD   I1  Ø           GET FIRST PARAMETER
      STO      PARA1
      LD   I1  1           GET SECOND PARAMETER
      STO      PARA2
      LD   I1  2           GET THIRD PARAMETER
      STO      PARA3
      MDX   1  3           SET UP TO RETURN TO MAINLINE
      STX   1  SUB         FOLLOWING THIRD PARAMETER
      LDX  I3  1Ø3         SET XR3 TO TRANSFER VECTOR
*                          IF ANY LIBF IS TO BE MADE
*                          WITHIN THE PROGRAM
*
*                          EXECUTION INSTRUCTIONS
*USE  STO  I1  Ø           TO STORE RESULT IN I, ETC.
*USE  STO  L   /FFFF       TO STORE RESULT IN FIRST
*                          WORD OF FORTRAN COMMON
*USE  LIBF     FLD         (OR ELD) AND
*     DC       DATA        TO STORE RESULT IN FAC
*
*                          EXIT FROM SUBROUTINE
XR1   LDX  L1  *-*         RELOAD INDEX REGISTERS
XR2   LDX  L2  *-*
XR3   LDX  L3  *-*
      LDD      TEMP        RELOAD A AND Q REGISTERS
      BSC  I   SUB         RETURN TO MAINLINE
TEMP  BSS  E   2
PARA1 DC       Ø
PARA2 DC       Ø
PARA3 DC       Ø
      END
```

LIBF SUBROUTINES

The source statements for subroutines that are called by a LIBF statement must be preceded by a LIBR statement.

At execution time, the LIBF call appears as a BSI instruction indexed by XR3 and with a displacement that reflects the transfer vector entry for the subroutine being called. XR3 contains the address of the transfer vector. The transfer vector entry contains a long BSI instruction to the subroutine entry point.

The following example illustrates a LIBF subroutine and shows how to define the entry point, save the machine status, get the address of the parameter list, and return to the calling program.



```
      HDNG     SAMPLE LIBF SUBROUTINE
*     LIBF     SUB1        SOURCE LANGUAGE
*     DC       PARA1       CALLING SEQUENCE
      LIBR                 LIBF CONTROL CARD
      ENT      SUB1        DEFINES ENTRY POINT
SUB1  DC       Ø           SUB1 ENTRY POINT
      STD      TEMP        SAVE MACHINE STATUS
      STX      XR1+1       THAT IS TO BE USED
      LDX  I1  SUB1        GET ADDRESS OF PARA LIST
      LD    1  -3          FROM TV
      STO      *+1
      LDX  L1  *-*         XR1 POINTS TO PARA LIST
      LD    1  Ø           GET PARAMETER
      STO      PARA1
      MDX   1  1           SET UP RETURN ADDR IN SUB1
      STX   1  SUB1
*
*                          SUBROUTINE OPERATION
XR1   LDX  L1  *-*         RESTORE MACHINE
      LDD      TEMP
      BSC  I   SUB1        RETURN TO MAINLINE
TEMP  BSS  E   2
PARA1 DC       Ø
      END
```

INPUT/OUTPUT SUBROUTINES

The procedures for writing input/output subroutines are similar to those for CALL or LIBF subroutines, except that an ISS statement is used to define the entry of the call section of the routine; also, the interrupt entry points must be defined.

The basic identification for the interrupt entry portion is the IAC code. There is a unique IAC code for each ILSW bit that is turned on by an I/O interrupt. At system generation time, the user defines the IAC codes and their corresponding ILSW bit. The same IAC code must be used when writing an I/O subroutine.

As stated previously, an ISS statement is used to define the call entry point (only one call entry point is permitted). If the subroutine is to be called by a LIBF statement, the ISS statement must be preceded by a LIBR statement. The LIBR statement is omitted if the subroutine is to be called by using a CALL statement (the CALL statement method must be used if the subroutine is to be called from a FORTRAN program). Following the ISS statement, there must be a pair of DC statements for each interrupt entry point. The first DC statement must define the IAC code for that entry, and the second DC must define the address of the interrupt entry point. This is followed by an ORG *-X where X is the number of DC statements.

The following is an example of how a typical ISS subroutine is written.

PROGRAM LISTING NO. 7: SAMPLE CARD I/O ROUTINE

```
                          HDNG    SAMPLE CARD I/O ROUTINE                 CRD00000
                      ************************************************** CRD00010
                      *               SAMPLE CARD I/O ROUTINE          * CRD00020
                      ************************************************** CRD00030
                      *                                                * CRD00040
                      *  THIS SUBROUTINE IS A SAMPLE OF THE TSX I/O    * CRD00050
                      *  ROUTINES.  IT IS CALLED VIA LIBF.  THE CALLING * CRD00060
                      *  SEQUENCE IS                                   * CRD00070
                      *                                                * CRD00080
                      *     LIBF    CARD                               * CRD00090
                      *     DC      /X00Y      CONTROL PARAMETER       * CRD00100
                      *     DC      AREA       I/O AREA ADDR           * CRD00110
                      *                                                * CRD00120
                      *  CONTROL PARAMETER IS COMPRISED OF 4 HEX DIGITS * CRD00130
                      *  OF WHICH ONLY X AND Y ARE USED.               * CRD00140
                      *                                                * CRD00150
                      *             X EQUAL 0  TEST FUNCTION-DO NOT    * CRD00160
                      *                             INCLUDE AREA PARA  * CRD00170
                      *             X EQUAL 1  READ FUNCTION           * CRD00180
                      *             X EQUAL 2  PUNCH FUNCTION          * CRD00190
                      *                                                * CRD00200
                      *             Y EQUAL 0  USE FIRST 1442          * CRD00210
                      *             Y EQUAL 1  USE SECOND 1442         * CRD00220
                      *                                                * CRD00230
                      *                                                * CRD00240
                      *  EXTERNAL REFERENCES TO FIXED AREA OF CORE     * CRD00250
                      *                                                * CRD00260
                      *     WORD    FUNCTION                           * CRD00270
                      *                                                * CRD00280
                      *     7       GENERAL I/O BUSY INDICATOR         * CRD00290
                      *     46      USER MASK IOCC LEVELS 0-13         * CRD00300
                      *     48      USER MASK IOCC LEVELS 14-23        * CRD00310
                      *     50      MASK ALL IOCC LEVELS 0-13          * CRD00320
                      *     52      MASK ALL IOCC LEVELS 14-23         * CRD00330
                      *     55      LOCATION WHERE TVSAV PUTS RET ADDR * CRD00340
                      *     90      I/O SUBROUTINE ENTRY TO MIC        * CRD00350
                      *     172     ENTRY POINT TO TVSAV               * CRD00360
                      *     173     ENTRY POINT TO TVEXT               * CRD00370
                      *                                                * CRD00380
                      *  NOTE THAT ON A PUNCH FUNCTION IT IS ASSUMED   * CRD00390
                      *     THAT THE USER HAS THE END BIT SET IN THE   * CRD00400
                      *     LAST WORD TO BE PUNCHED.                   * CRD00410
                      *                                                * CRD00420
                      ************************************************** CRD00430
                      *                                                  CRD00440
                          LIBR                 SIGNIFIES THIS IS A LIBF  CRD00450
0000      03059100        ISS   2 CARD         2 IS  THE NO. OF INT      CRD00460
                      *                              ENTRY POINTS        CRD00470
                      *                              AND CARD IS LIBF    CRD00480
                      *                              ENTRY POINT         CRD00490
0000 0  0002              DC      2            1442-1 IAC CODE           CRD00500
0001 1  003C              DC      INT1         1442-1 INT ENTRY POINT    CRD00510
0002 0  0011              DC      17           1442-2 IAC CODE           CRD00520
0003 1  003F              DC      INT2         1442-2 INT ENTRY POINT    CRD00530
0004                      ORG     *-4          CAUSES OVERLAY OF LD INFO CRD00540
                      *                                                  CRD00550
                      *   CALL SECTION OF SUBROUTINE                     CRD00560
                      *                                                  CRD00570
0000 0  0000          CARD DC     0            LIBF ENTRY POINT          CRD00580
0001 00 448000AC          BSI  I  172          CALL TVSAV  TO SAVE MACH- CRD00590
                      *                        INE REGISTERS AND STATUS. CRD00600
                      *                        ALSO SETS WORD 55 TO POINT CRD00610
                      *                        TO FIRST PARAMETER--RETURN CRD00620
                      *                        ADDRESS.                  CRD00630
0003 00 65800037          LDX  I1 55           XR1#LIBF PARAMETERS       CRD00640
0005 01 6600005A          LDX  L2 CD1          XR2#1442-1 DEVICE TABLE   CRD00650
0007 0  C100              LD   X1 0            DETERMINE 1442 TO BE USED CRD00660
0008 0  4804              BSC     E            SKIP IF FIRST 1442        CRD00670
0009 0  7208              MDX   2 CD2-CD1      INCREMENT TO POINT TO CD2 CRD00680
000A 0  180C              SRA     12           IS THE FUNCTION A TEST    CRD00690
000B 01 4C200012          BSC  L  CONTN,Z      BRANCH IF NOT A TEST      CRD00700
000D 0  C204              LD   X2 BUSY         GET BUSY INDICATOR        CRD00710
000E 0  4818              BSC     +-           SKIP IF ON                CRD00720
000F 0  7101              MDX   1 1            INCREMENT RET ADDR TO SKIP CRD00730
0010 0  7101              MDX   1 1            TWO WORDS ON RET TO USER  CRD00740
0011 0  7025              MDX     OUT          BRANCH TO SET UP EXIT     CRD00750
                      *                                                  CRD00760
                      *   FUNCTION IS NOW A READ OR PUNCH                CRD00770
0012 00 0C000032      CONTN XIO L 50           MASK LEVELS 0-13          CRD00780
0014 00 0C000034          XIO  L  52           MASK LEVELS 14-23         CRD00790
0016 0  C204              LD   X2 BUSY         GET BUSY INDICATOR        CRD00800
0017 01 4C18001F          BSC  L  READY,+-     BRANCH IF IT IS TURNED OFF CRD00810
0019 00 0C00002E          XIO  L  46           UNMASK TO USER MASK TO    CRD00820
001B 00 0C000030          XIO  L  48           ALLOW CARD OP-COMPLETE    CRD00830
001D 0  1000              NOP                                            CRD00840
001E 0  70F3              MDX     CONTN        SEE IF ROUTINE STILL BUSY CRD00850
001F 0  0A04          READY XIO X2 SENSE       LOOP UNTIL  1442 IS IN A  CRD00860
```

```
0020 01 4C04001F            BSC L  READY,E    READY CONDITION            CRD00870
0022 0  C000                LD     *          TURN BUSY INDICATOR ON     CRD00880
0023 0  D204                STO X2 BUSY       TO INDICATE DEVICE BEING   CRD00890
                     *                        USED                       CRD00900
0024 0  C100                LD  X1 0          GET FUNCTION PARAMETER     CRD00910
0025 0  180C                SRA    12         TEST FOR READ OR PUNCH     CRD00920
0026 01 4C040030            BSC L  READ,E     BRANCH IF FUNCTION IS READ CRD00930
                     *     FUNCTION IS PUNCH                             CRD00940
0028 0  C101         PUNCH LD  X1 1          GET SECOND PARAMETER       CRD00950
0029 0  D202                STO X2 PH         PUT AREA PARA IN PH IOCC   CRD00960
002A 0  0A02                XIO X2 PH         START PUNCHING A CARD      CRD00970
002B 00 74010007            MDX L  7,1        INCREMENT GEN I/O BUSY IND CRD00980
002D 0  1000                NOP                                          CRD00990
002E 0  7101                MDX  1 1          INCREMENT RET ADDR         CRD01000
002F 0  7007                MDX    OUT        GO TO EXIT                 CRD01010
                     *     FUNCTION IS A READ                            CRD01020
0030 0  C101         READ  LD  X1 1          GET SECOND PARAMETER       CRD01030
0031 0  D200                STO X2 RD         PUT AREA ADDR IN RD IOCC   CRD01040
0032 0  0A00                XIO X2 RD         START READING A CARD       CRD01050
0033 00 74010007            MDX L  7,1        INCREMENT GEN I/O BUSY IND CRD01060
0035 0  1000                NOP                                          CRD01070
0036 0  7101                MDX  1 1          INCREMENT RET ADDR         CRD01080
                     *     SETUP TO RETURN TO USER                       CRD01090
0037 0  7101         OUT   MDX  1 1          INCREMENT RET ADDR         CRD01100
0038 00 6E000037            STX L2 55         55 NOW CONTAINS THE RETURN CRD01110
                     *                        ADDRESS TO USER            CRD01120
003A 00 448000AD            BSI I  173        RETURN TO USER VIA TVEXT   CRD01130
                     *                                                   CRD01140
                     *     INTERRUPT SECTION OF SUBROUTINE               CRD01150
                     *                                                   CRD01160
003C 01 6600005A     INT1  LDX L2 CD1        XR2#1442-1. THIS IS        CRD01170
003E 0  7002                MDX    *+2        INTERRUPT ENTRY POINT FOR  CRD01180
                     *                        1442-1                     CRD01190
003F 01 66000062     INT2  LDX L2 CD2        XR2#1442-2. THIS IS        CRD01200
                     *                        INTERRUPT ENTRY POINT FOR  CRD01210
                     *                        1442-2                     CRD01220
0041 0  0A06                XIO X2 SENSR      SENSE DSW WITH RESET       CRD01230
0042 0  1002                SLA    2          TEST FOR ERROR             CRD01240
0043 01 4C100051            BSC L  DONE,-     BRANCH IF NO ERROR         CRD01250
0045 0  3000                WAIT              WAIT FOR USER TO RELOAD CD CRD01260
0046 0  0A04         READX XIO X2 SENSE      LOOP UNTIL  1442 IS IN A   CRD01270
0047 01 4C040046            BSC L  READX,E    READY CONDITION            CRD01280
0049 0  C200                LD  X2 RD         TEST FOR LAST FUN A RD     CRD01290
004A 01 4C18004E            BSC L  REDOP,+-   BRANCH IF NOT READ         CRD01300
004C 0  0A00                XIO X2 RD         REDO READ                  CRD01310
004D 0  7001                MDX    *+1        EXIT TO MIC                CRD01320
004E 0  0A02         REDOP XIO X2 PH         REDO PUNCH                 CRD01330
004F 00 4C80005A            BSC I  90         RETURN TO MIC              CRD01340
0051 0  1010         DONE  SLA    16         CLEAR ALL INDICATORS       CRD01350
0052 0  D200                STO X2 RD         CLEAR READ IOCC            CRD01360
0053 0  D202                STO X2 PH         CLEAR PUNCH IOCC           CRD01370
0054 0  D204                STO X2 BUSY       CLEAR BUSY INDICATOR       CRD01380
0055 00 74FF0007            MDX L  7,-1       DECREMENT GEN I/O BUSY IND CRD01390
0057 0  1000                NOP                                          CRD01400
0058 00 4C80005A            BSC I  90         RETURN TO MIC              CRD01410
                     *                                                   CRD01420
                     *     DEVICE TABLES FOR 1442-1 AND 1442-2           CRD01430
                     *                                                   CRD01440
                     *     NOTE THAT THE NO. IN COLUNM 71 IS THE         CRD01450
                     *     DISPLACEMENT OF THAT WORD FROM THE START      CRD01460
                     *     OF THE DEVICE TABLE                           CRD01470
005A    0000                BSS E  0          DEVICE TABLE MUST BEGIN ON CRD01480
                     *                        AN EVEN ADDR BECAUSE OF    CRD01490
                     *                        IOCC.                      CRD01500
005A 0  0000         CD1   DC     0          DEVICE TABLE FOR 1442-1  0 CRD01510
005B 0  1600                DC     /1600      READ IOCC                1 CRD01520
005C 0  0000                DC     0                                   2 CRD01530
005D 0  1500                DC     /1500      PUNCH IOCC               3 CRD01540
005E 0  0000                DC     0          BUSY INDICATOR           4 CRD01550
005F 0  1700                DC     /1700      SENSE IOCC               5 CRD01560
0060 0  0000                DC     0                                   6 CRD01570
0061 0  1701                DC     /1701      SENSE/RESET IOCC         7 CRD01580
0062    0000                BSS E  0                                     CRD01590
0062 0  0000         CD2   DC     0          DEVICE TABLE FOR 1442-2  0 CRD01600
0063 0  8E00                DC     /8E00      READ IOCC                1 CRD01610
0064 0  0000                DC     0                                   2 CRD01620
0065 0  8D00                DC     /8D00      PUNCH IOCC               3 CRD01630
0066 0  0000                DC     0          BUSY INDICATOR           4 CRD01640
0067 0  8F00                DC     /8F00      SENSE IOCC               5 CRD01650
0068 0  0000                DC     0                                   6 CRD01660
0069 0  8F01                DC     /8F01      SENSE/RESET IOCC         7 CRD01670
                     *                                                   CRD01680
                     *     DEVICE TABLE EQUATES                          CRD01690
                     *                                                   CRD01700
0000                 RD    EQU    0          READ IOCC                  CRD01710
0002                 PH    EQU    2          PUNCH IOCC                 CRD01720
0004                 BUSY  EQU    4          BUSY INDICATOR             CRD01730
0004                 SENSE EQU    4          SENSE IOCC                 CRD01740
0006                 SENSR EQU    6          SENSE/RESET IOCC           CRD01750
006A                       END                                           CRD01760
```

PROGRAMMING SUBROUTINES USING REENTRANT CODING

NEED FOR REENTRANT CODING

One of the basic problems that arises in multi-level programming is requirement of the same subroutine by different levels of operation.

For example, the computer is servicing a mainline program which is executing a square-root subroutine when an external interrupt occurs. The hardware interrupt will automatically branch to an address which will allow servicing of the interrupt.

The program that services the interrupt may also require use of a square-root subroutine. If a method of reentrant coding were not used, the identical square-root subroutine would have to be in core storage twice (once for each program that called it); otherwise, the intermediate results which are needed when the computer returns to complete the mainline program would be destroyed by the interrupt program.

CONCEPT OF LEVEL WORK AREAS

To allow one subroutine to be entered at any time and from any interrupt level, without loss of intermediate results, a method of reentrant coding using level work areas is used.

Reentrant coding is defined as coding which allows a program to be entered and executed from different levels without destroying the intermediate results.

The IBM 1800 TSX System provides features which facilitate the coding of reentrant subroutines.

Each interrupt level specified by the user is provided with a level work area of 104 locations, which are reserved for the exclusive use of programs operating on that priority level.

The first 62 of these locations are reserved for specific routines (MIC, QZSAV, etc.) while the remaining 42 locations are available to allow other subroutines (arithmetic, functional, etc.), to maintain their ability to reenter.

The start address of the level work area for any priority level always appears in location LWA (fixed location $104_{10} = 68_{16}$). If an index register is loaded with the contents of this location, and all references to temporary storage locations are indexed, 42 temporary storage locations are made available to the subroutine for each level it may be operating on. If the subroutine is reentered, different effective addresses are generated for each such indexed operand, and the reentry problem is solved.

The following sequence of instructions illustrates how the contents of the A-register are saved in TEMP in the level work area and later restored by the instruction at LOAD:

Label	Operation	F	T	Operands & Remarks
21 25	27 30	32	33	35 40 45 50 55 60 65 70
LWA	EQU			104
TEMP	EQU			58
	LDX	I	1	LWA WORK AREA ADDR TO XR1
STRE	STO		1	TEMP SAVE A-REGISTER IN TEMP
	•			
	•			
	•			
LOAD	LD		1	TEMP RESTORE A-REGISTER

Should the subroutine be interrupted and reentered, there will be no storage conflicts, since the contents of LWA changes with each interrupt level. Hence, the instructions at STRE and LOAD reference different effective addresses for each interrupt level.

MECHANISM FOR REENTRANT CONTROL

For each interrupt serviced, MIC (Master Interrupt Control program) saves and subsequently restores the contents of the A- and Q-registers, index registers, machine status, and locations WK4 ($54_{10} = 36_{16}$) and WK5 ($55_{10} = 37_{16}$). MIC also sets LWA to the correct level work area address for each interrupt level.

Since locations WK4 and WK5 are saved by MIC for each interrupt level, these locations may also be used for temporary storage by reentrant subroutines, e.g., loading and storing of index registers. Furthermore, these locations are also used for other purposes, as explained below.

Protecting Entry and Return Addresses

The first location of a callable subroutine is set by a BSI instruction. As with all fixed locations upon reentry, this location may be changed and the return address may be lost. The TSX System supplies two pairs of subroutines which provide a method of protecting the return address. They also perform several additional functions useful for subroutines.

Subroutines Referenced by a CALL Instruction

Label 21–25	Operation 27–30	F 32	T 33	Operands & Remarks 35–65
QZSAV	EQU			154
QZEXT	EQU			155
SUBRT	DC			Ø ENTRY TO CALL ROUTINE
	BSI	I		QZSAV CALL TO QZSAV
	•			
	•			
	•			
EXIT	BSI	I		QZEXT EXIT FROM SUBROUTINE

The QZSAV subroutine saves the contents of index registers 1, 2, and 3, the A- and Q-registers and machine status and places the return address in location WK4 ($54_{10} = 36_{16}$). In addition, index registers 1 and 3 are set to the first location of the level work area, and index register 2 is set to $127_{10} = 7F_{16}$.

The QZEXT subroutine restores the index registers, machine status, and A- and Q-registers and returns control to the calling routine via a BSC I WK4. The address set in WK4 by QZSAV must, therefore, be incremented by 1 for every parameter following the CALL.

Subroutines Referenced by a LIBF Instruction

For subroutines referenced by a LIBF (1-word BSI) instruction through the transfer vector:

Label 21–25	Operation 27–30	F 32	T 33	Operands & Remarks 35–65
TVSAV	EQU			172
TVEXT	EQU			173
SUBRT	DC			Ø ENTRY TO LIBF ROUTINE
	BSI	I		TVSAV CALL TO TVSAV
	•			
	•			
	•			
EXIT	BSI	I		TVEXT EXIT FROM SUBROUTINE

The TVSAV subroutine saves the contents of index registers 1 and 2, the A- and Q-registers, and machine status and places the return address in WK5 ($55_{10} = 37_{16}$). In addition, index registers 1 and 3 are set to the first location of the level work area, and index register 2 is set to $127_{10} = 7F_{16}$.

The TVEXT subroutine restores the contents of index registers 1 and 2, A- and Q-registers, and machine status. Index register 3 is set to the transfer vector location, and control is returned to the calling routine via a BSC I WK5. The address set in WK5 by TVSAV must, therefore, be incremented by 1 for every parameter following the LIBF.

Other Considerations

It should be understood that the use of QZSAV or TVSAV does not obviate careful logic control. If parameters follow the call to the subroutine, it is the responsibility of the subroutine to obtain these parameters and to adjust WK4 or WK5.

If subroutines are nested, that is, one subroutine calls another, care must be exercised to save and restore the storage locations used by QZSAV and TVSAV across the nested call, as well as the return address in WK4 or WK5. Furthermore, nested subroutines must be planned so that the same locations in the level work area are not used by more than one subroutine.

Note that TVSAV, TVEXT, QZSAV, and QZEXT are referenced by indirect BSI instructions and <u>not</u> by CALL statements. The call to TVSAV or QZSAV must be the first instruction executed in (and immediately following) the entry location, as illustrated.

MASKING OUT THE INTERRUPTS

Another method of providing reentrant coding is to prevent the interrupts from being recognized.

In Assembler language, it is possible to use the XIO command with the IOCC-masking words provided by the TSX system. To mask all interrupt levels completely, the following instructions may be executed:

Label 21–25	Operation 27–30	F 32	T 33	Operands & Remarks 35–65
MSK1	EQU			5Ø LOCATIONS OF
MSK2	EQU			52 MASK IOCC WORDS
	XIO	L		MSK1 MASK ALL INTERRUPT
	XIO	L		MSK2 LEVELS

To restore the interrupt mask status, the following instructions may be executed:

Label 21–25	Operation 27–30	F 32	T 33	Operands & Remarks 35–65
MSK3	EQU			46 LOCATIONS OF UNMASK
MSK4	EQU			48 IOCC WORDS
	XIO	L		MSK3 RESTORE INTERRUPT
	XIO	L		MSK4 STATUS

This particular method of reentrant coding is effective and permissible, but is, in general, undesirable. If interrupts are prevented from being recognized as may occur, the philosophy of the IBM 1800 interrupt system is defeated. However, for short sequences of instructions, the method of masking out the interrupts may be the fastest means of obtaining reentrant coding.

PROGRAMMING NOTES

The following examples illustrate the different approaches that may be used to write reentrant subroutines, and to explain the need for WK4 and WK5 (words 54 and 55 of the Fixed Area in core storage). These words are saved by MIC in the same manner that the accumulator and certain index registers are preserved during the handling of an interrupt.

Both examples depict a method of storing what is contained at the effective address reached by an index register plus its displacement, loading that value into another index register, and be reentrant.

```
Label  Operation  F T  Operands & Remarks
21  25 27    30  32 33 35   40   45   50   55   60   65
* NON REENTRANT CASE

CONST  EQU          53
        .
        .
        .
       LD       1  CONST
       STO         *+1
       LDX     L2  *-*

* REENTRANT APPROACH #1

CONST  EQU          53
MSK1   EQU          50
MSK2   EQU          52
MSK3   EQU          46
MSK4   EQU          48
        .
        .
        .
       XIO     L   MSK1          MASK ALL INTERRUPT
       XIO     L   MSK2          LEVELS
       LD       1  CONST
       STO         *+1
       LDX     L2  *-*
       XIO     L   MSK3          UNMASK TO USERS
       XIO     L   MSK4          MASK STATUS
* REENTRANT APPROACH #2

CONST  EQU          53
        .
        .
        .
       LD       1  CONST
       STO     L   54            LOCATION OF WK4
       LDX     I2  54

```

WRITING USER-PROGRAMS FOR EXECUTION UNDER THE TASK ABSOLUTE LOADER

The TASK Absolute Loader can be used to load programs from cards to core for execution under TASK or for the storing on disk of user-written programs or data. The TSX system must be in the off-line (nonprocess) mode. To call the absolute loader, load or restart TASK and set sense switch 0 on.

PROGRAM/DATA FORMAT

The User programs and data must be assembled absolute and origined above the last address of TASK. The object deck must be of the relocatable format type, i.e., compressor output, not core image. The execution address on the end-of-program card must be the address of the first user instruction if the program is to be executed, and must be the address of the program word count if the program is to be stored on disk. No LIBFs or CALLs are allowed in the program, but any TASK subroutine (DISKN, TYPEN, etc.) can be called as defined in TASK I/O Subroutines. The TASK object program set is an example of programs designed to run under the absolute loader.

NOTE: When executing user-written absolute programs under TASK, it is best to use an off-line cartridge, or a cartridge that does not contain the TSX system, so that TSX system areas which are not file protected are not destroyed. For example, TASK uses sectors 05C0 and up for buffering of 1053 messages. The upper limit of this area was established by the user at TASK assembly time.

The source deck format for executable programs is

```
         ABS              ABC is a core
         ORG       ABC    address above the
START    LD               last TASK address
         .                (see NOTE below)
         .
         .
         User's
         Program
         .
         .
         .
         END       START
```

The source deck format for data or programs stored on disk is

```
         ABS              ABC is a core
         ORG       ABC    address above the
                          last TASK address
                          (see NOTE below)
```

```
START   DC      A-START-2
        DC      SECAD      SECAD is the
                           first sector ad-
                           dress where the
                           program is to
                           be stored.
        .
        .
        .
        User's
        Program/
        Data
        .
        .
        .
A       EQU     *
        END     START
```

NOTE: To insure that the program is always above the end of TASK, let ABC be greater than or equal to /F1F0. The highest address of the program must be less than or equal to /FFFF.

It is the user's responsibility to recall the data or program from the disk.

ABSOLUTE LOADER OPERATION

When TASK has been reloaded or restarted, the following message is printed.

```
TASK 1800 TSX
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER
```

1. Set Sense switch 0 on.
2. Place the program to be loaded in the card read punch hopper. A stacked input is allowable.
3. Press reader START.
4. The user now has the option of selecting manual or automatic mode before pressing Console START.

Manual Mode

If data switch 15 is off, the absolute loader operates in the manual mode. After the program has been loaded to core, the following message is printed.

```
DATA SW 0 ON LD DISK OFF EXECUTE
```

Set data switch 0 off and press console START to execute the program just loaded. Set data switch 0 on and press Console START to write the program or data to disk. If the program or data is written to disk, the absolute loader starts reading the next program in the card reader into core after performing the disk write function.

Automatic Mode

If data switch 15 is on, the absolute loader operates in the automatic mode. After loading the user's program, the absolute loader executes it unless the control card illustrated below has been placed in front of the user's program. The format of the control card is:

column 1 = 8
column 2 = +
column 4 = 9
column 13 = 1
all other columns = blank

The user can assemble this control card in his object deck by placing the two source cards shown below right after the ABS card in his source deck.

```
ORG     40
DC       1
```

If the program is loaded in automatic mode and the control card has been included, the program will be stored on disk and the next program in the reader will be loaded.

The following (Program Listing No. 8) is an example of the loading and execution of an absolute 80-80 program using the TASK Absolute Loader. Note that the program is assigned for the upper 4K of core. The ABS card is used to indicate that this is an absolute assembly. The start address is present in the END statement.

PROGRAM LISTING NO. 8: TASK ABSOLUTE PROGRAM FOR DOING AN 80-80 LIST OF CARDS ON THE LIST PRINTER

```
// JOB
// ASM LIST
  *LIST
  *PUNCH
  *PRINT SYMBOL TABLE
                              ABS
                         ****************************************************
                         *                                                  *
                         *      TASK ABSOLUTE PROGRAM FOR DOING AN 80-80    *
                         *              LIST OF CARDS ON THE  LIST          *
                         *                       PRINTER.                   *
                         *                                                  *
                         ****************************************************
0000                          ORG     /F1F0       PLACE IN UPPER 4K
007D                     LSTPT EQU    125         LIST PRINTER ENTRY POINT
003D                     CARDN EQU    61          CARDN ENTRY POINT
003A                     HOLEB EQU    58          HOLEB ENTRY POINT
003B                     EBPRT EQU    59          EBPRT ENTRY POINT
00A6                     $LORG EQU    166         LIST PT DEVICE TYPE IND
0020                     CIND  EQU    32          CD IND ON INT LEV WK AREA
0068                     $TVWK EQU    104         INT LEV WK AREA LOC ADDRES
                         *
F1F0 00 67800068         START LDX  I3 $TVWK      TELL CARDN NOT TO CHECK
F1F2 0  C000                   LD      *             FOR //  CARDS
F1F3 0  D320                   STO  X3 CIND
F1F4 00 C40000A6               LD   L  $LORG      WHAT DEVICE IS LIST PRINTR
F1F6 00 4C08F1FB               BSC  L  LOOP,+     BRANCH IF 1053
F1F8 00 4480007D               BSI  I  LSTPT      SKIP TO CHANNEL 1
F1FA 0  3100                   DC      /3100
F1FB 00 4480003D         LOOP  BSI  I  CARDN      READ A CARD
F1FD 0  1000                   DC      /1000
F1FE 0  F291                   DC      IAREA
F1FF 0  0000                   DC      0
F200 00 4480003D               BSI  I  CARDN      LOOP BUSY
F202 0  0000                   DC      0
F203 0  70FC                   MDX     *-4
F204 00 4480003A               BSI  I  HOLEB      CONVERT TO EBC CODE
F206 0  0000                   DC      0
F207 0  F292                   DC      IAREA+1
F208 0  F292                   DC      IAREA+1
F209 0  0050                   DC      80
F20A 00 C40000A6               LD   L  $LORG
F20C 00 4C20F24F               BSC  L  P1443,Z    BRANCH IF 1443
F20E 00 4480007D         P1053 BSI  I  LSTPT      LOOP BUSY
F210 0  0000                   DC      0
F211 0  70FC                   MDX     *-4
F212 00 4480007D               BSI  I  LSTPT      RETURN CARRIER
F214 0  2000                   DC      /2000
F215 0  F28F                   DC      RETUR
F216 0  0000                   DC      0
F217 0  6114                   LDX   1 20         SET UP TO SUPRESS TRAILING
F218 0  6947                   STX   1 OAREA      SET UP WDCT
F219 00 C500F2A5         LP1   LD   L1 IAREA+20   BLANKS--TEST FOR BLANK
F21B 0  F043                   EOR     H4040
F21C 00 4C20F22A               BSC  L  NZ1,Z      BRANCH IF NOT BLANK
F21E 0  71FF                   MDX   1 -1         DECREMENT WDCT
F21F 0  70F9                   MDX     LP1        TEST NEXT CHARACTER
F220 0  6954                   STX   1 OAREA+21   SET UP ZERO WDCT
F221 0  6114                   LDX   1 20         TEST FIRST HALF OF CARD
F222 00 C500F291         LP2   LD   L1 IAREA      TEST FOR BLANK
F224 0  F03A                   EOR     H4040
F225 00 4C20F22C               BSC  L  NZ2,Z      BRANCH NOT BLANK
F227 0  71FF                   MDX   1 -1
F228 0  70F9                   MDX     LP2        TEST NEXT CHARACTER
F229 0  70D1                   MDX     LOOP       ALL OF CARD IS BLANK
F22A 0  694A             NZ1   STX   1 OAREA+21
F22B 0  7001                   MDX     *+1
F22C 0  6933             NZ2   STX   1 OAREA
F22D 00 4480003B               BSI  I  EBPRT      SET UP FIRST HALF OF MESS
F22F 0  0001                   DC      1            TO BE PRINTED
F230 0  F292                   DC      IAREA+1
F231 0  F261                   DC      OAREA+1
F232 0  0028                   DC      40
F233 00 4480007D               BSI  I  LSTPT      LOOP BUSY
F235 0  0000                   DC      0
F236 0  70FC                   MDX     *-4
F237 00 4480007D               BSI  I  LSTPT      PRINT FIRST HALF OF CARD
F239 0  2000                   DC      /2000
F23A 0  F260                   DC      OAREA
F23B 0  0000                   DC      0
F23C 0  C038                   LD      OAREA+21   TEST SECOND HALF FOR BLANK
F23D 00 4C08F1FB               BSC  L  LOOP,+     BRANCH IF BLANK
F23F 00 4480003B               BSI  I  EBPRT      SET UP SECOND HALF OF CARD
F241 0  0001                   DC      1            TO BE PRINTED
F242 0  F2A6                   DC      IAREA+21
F243 0  F276                   DC      OAREA+22
F244 0  0028                   DC      40
```

```
F245 00 4480007D          BSI  I  LSTPT        LOOP BUSY
F247 0  0000              DC      0
F248 0  70FC              MDX     *-4
F249 00 4480007D          BSI  I  LSTPT        PRINT SECOND OF CARD
F24B 0  2000              DC      /2000
F24C 0  F275              DC      OAREA+21
F24D 0  0000              DC      0
F24E 0  70AC              MDX     LOOP         GO READ NEXT CARD
F24F 00 4480007D    P1443 BSI  I  LSTPT        LOOP BUFFER BUSY
F251 0  0010              DC      /0010
F252 0  70FC              MDX     *-4
F253 00 4480003B          BSI  I  EBPRT        SET UP CARD TO BE PRINTED
F255 0  0001              DC      1
F256 0  F292              DC      IAREA+1
F257 0  F266              DC      OAREA+6
F258 0  0050              DC      80
F259 00 4480007D          BSI  I  LSTPT        PRINT THE CARD
F25B 0  2100              DC      /2100
F25C 0  F260              DC      OAREA
F25D 0  0000              DC      0
F25E 0  709C              MDX     LOOP         GO READ NEXT CARD
                    *
                    *     TABLE AREA
                    *
F25F 0  4040        H4040 DC      /4040        EBC BLANK
F260 0  002D        OAREA DC      45           OUTPUT BUFFER
F261 0  0000              DC      0
F262 0  0000              DC      0
F263 0  0000              DC      0
F264 0  0000              DC      0
F265 0  0000              DC      0
F266    000F              BSS     15
F275 0  0014              DC      20
F276    0019              BSS     25
F28F 0  0001        RETUR DC      1            RETURN CARRIER MESSAGE
F290 0  8121              DC      /8121
F291 0  0050        IAREA DC      80           INPUT BUFFER
F292    0050              BSS     80
F2E2    F1F0              END     START
                    *
                    *            SYMBOL TABLE
                    *

       CARDN 003D   CIND  0020   EBPRT 003B   HOLEB 003A   H4040 F25F
       IAREA F291   LOOP  F1FB   LP1   F219   LP2   F222   LSTPT 007D
       NZ1   F22A   NZ2   F22C   OAREA F260   P1053 F20E   P1443 F24F
       RETUR F28F   $LORG 00A6   $TVWK 0068   START F1F0

       NO ERRORS IN ABOVE ASSEMBLY.
LIST
DUP FUNCTION COMPLETED
```

BASIC CONCEPTS OF DATA ACQUISITION AND PROCESS CONTROL SYSTEMS (DACS)

INTRODUCTION

Data Acquisition and Process Control Systems are by definition real-time systems. In real-time processing, inputs may arrive randomly from the process being monitored to the computer which rapidly responds to each input, usually by transmitting an output back to the process. This is in contrast to conventional batch processing where groups of inputs are processed by passes through the computer. The notion of real-time usually implies that a computer is responding to inputs as they occur in the physical world.

The principal functions of a real-time data acquisition and process control computer system are: data scanning, data logging, process calculations, process outputs, input-output control, operator-machine communication, and specialized system monitoring. Each of these functions is implemented by one or more programs permanently stored in core or in secondary storage.

In general, data logging/data acquisition applications are basically monitor systems where the data signal traffic is toward the computer process I/O. Control systems (supervisory, operator-guide, or direct digital control), on the other hand, are characterized by a two-way signal flow across the process/computer interface. For the most part, data acquisition implies that the process operates in its normal manner without being affected by the computer, whereas computer control systems imply that the process is directly controlled by actions and commands of the computer.

Implementation of data acquisition and process control systems in real-time requires certain process I/O hardware which significantly affects software requirements. Most important of these are the analog input and output hardware which require relay and possibly solid-state multiplexing devices together with a multi-level priority interrupt system.

Analog Process I/O

By the very nature of processes, many internal process signals are analog -- that is, continuous functions with respect to time. Many instruments have been developed to pick-off various signals of interest and to modify them (that is, amplify, filter, linearize, etc.) in order to provide data or control signals. Analog-to-digital converters (ADC) are used to convert the resultant analog signals for computer entry. The ADC and amplifiers in a DACS may also operate directly on the output of a signal transducer.

Transducers have been designed around physical laws to convert one form of energy to another. Transducers used with data acquisition and process control systems usually convert the various physical quantities to be measured to their voltage analogs. For example, transducers are available to convert pressure to voltage and temperature to voltage. The quality of the transducer is determined by the accuracy, repeatability, linearity, and proportional range of the parameter to voltage conversion. In almost every process, temperature is one of the variables of interest, and in many cases, thermocouples are used to measure temperature. Strain gauges may be used to measure pressure, deformation, and loading. Thus, depending upon the size and package, one could use a strain gauge to measure direct blood pressure or to measure stresses set up in a missile structural member when the missile engine is ignited. Similarly, a thermocouple could be used to measure a blast furnace process temperature or air temperature.

To convert the transducer-produced voltage to digital values, an analog-to-digital converter (ADC) is used. Since an ADC can operate at a high rate compared to the rate of change of the individual signal voltages and because ADC units are relatively expensive, it is customary to time-share the ADC among a number of input voltages through an analog multiplexer.

An analog multiplexer is a device which switches the various analog inputs to the ADC. Most multiplexers can be programmed for sequential scan where each input channel is in turn scanned and connected for conversion to the ADC. Random scan sequences can also be programmed. Random scanning, as the name implies, permits any arbitrary sequence of input channels to be scanned and connected in turn to the ADC.

In summary, the signal flow in the analog process I/O can be explained as follows. Multiple parameters are continuously converted to voltages by appropriate transducers and signal conditioning electronics. The resultant voltages form inputs to a multiplexer. Under command of the processor-controller, multiplex switches are closed to allow an analog input signal to be amplified (in most cases) by a time-shared amplifier. The ADC connected to the amplifier output does the actual voltage to digital conversion, with the resultant digital value being inputted to the computer. In this manner, the signals of interest in the process are sampled, measured and entered into the computer for additional processing.

Data flow from the computer to the process follows a converse procedure. Digital data is entered into multiple registers by the computer. Each register is connected to a digital-to-analog converter (DAC). A DAC is basically a digitally-controlled voltage or current source whose output is the voltage analog of the digital value. The analog outputs in turn control various set-points and other control points in the system. In some systems, a single DAC is connected to analog memory devices via an analog multiplexer. In this way, a single DAC can be used to provide many output signals.

By using digital-to-analog converters and analog-to-digital converters, a digital computer can be made to communicate with process signals and control equipment. It should be noted that the digital process I/O and the analog process I/O are all under program control.

Digital Process I/O

In addition to the analog parameters which must be monitored and controlled, there are other process signals which are binary in nature. Thus, for example, relays can be open or closed, a voltage level may or may not be present, or a parameter may be represented as a sequence of pulses. The computer must be able to accept this data which is essentially binary in form. Similarly, the computer must be capable of producing binary outputs to control devices such as relays. The subsystem in a DACS, which handles these types of signals, is generally referred to as digital process I/O.

Two of the process I/O functions provided on the IBM 1800 Data Acquisition and Control System are contact sense and voltage sense. The output of the contact sense circuit signifies when a contact closure has occurred on the input. The output of the voltage sense circuit indicates when an input signal has exceeded a preset level. The output of each circuit is connected to a single bit in a register. The processor-controller can scan these registers at rates from 100, 000 up to 500, 000 words (groups) per second.

Closely allied with these digital input features is process interrupt. Process interrupt is a vital feature for real-time control because the computer is basically a sequential machine. When a significant process event such as an alarm occurs, as indicated by a relay closure or voltage level, a signal is transmitted to the computer as an interrupt requiring a special subroutine to take appropriate action. Interrupts are usually assigned in order of priority, so that if two occur at once, the more important is serviced first by the computer. In essence, process interrupt is merely a special form of voltage/contact sense. The feature of customer-assignable multi-priority interrupts clearly differentiates a process control computer from the normal data processing system.

Pulse inputs form another category of digital inputs. A typical process source is the turbine flow meter which generates pulses at varying rates from a few pulses per second up to several thousand pulses per second. Electronic counters are used to accumulate the pulses; the response to an interrupt signal on counter overflow or a periodic scanning of the counter register then accomplishes transfer of the accumulated count data into the computer. High speed pulse counters and scalers for megacycle rates are also adaptable for high-speed data acquisition applications.

Similarly, some control devices in a typical process environment require input data in the form of pulses. Several types of output are available, one form being pulse output. The primary purpose of this output is to provide for pulse trains to operate such devices as latches, set point indicators, and other stepping motor devices.

One form of output which is opposite to the contact sense feature is Electronic Contact Operate (ECO). Many of the required control and display operations in a typical process application involve binary action; that is, a piece of equipment is turned on or off, or a valve is open or shut completely. This capability is provided by ECO.

To increase the versatility of the data acquisition and process control system, an output feature is provided whereby a digital word can be transferred to an external piece of equipment. The register output feature provides this capability on the 1800. Examples of its use might include transfer of data to another computer or to a display device.

Process-input-data flow to the computer is illustrated in Figure 89, as accomplished on an IBM 1800 Data Acquisition and Control System, together with corresponding process output features. Digital data may be read in under direct program control, as the program executes read-in instructions, or additional channel controls can be employed to sequence data to previously assigned areas of core storage without disrupting normal program operation. This is known as cycle-steal. When data is ready for entry, a cycle-steal operation is initiated by the channel control circuitry. One memory cycle of computer operation is used to read-in the data word directly to the assigned core storage location.

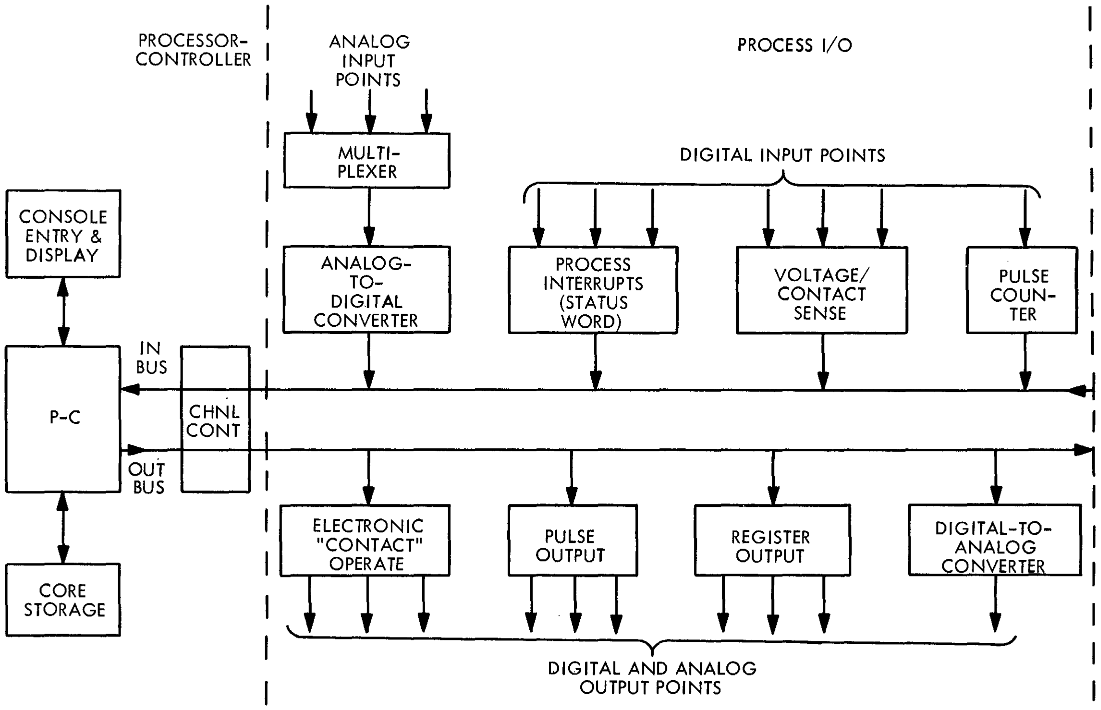


Figure 89. IBM 1800 Data Acquisition and Control System

DATA ACQUISITION SYSTEMS

Data acquisition/data logging systems are generally the simplest form of DACS. When used in a data acquisition role, the computer controls the real-time collection of data from the process. Many data acquisition systems operate on analog signals recorded on magnetic tape rather than in real-time. However, the problems involved in both situations are similar, except of course, in real-time only one opportunity to capture the data exists. To overcome this, an analog recording is sometimes made simultaneously for backup.

In data acquisition/data logging applications, the data flows from the process to the computer system (or analog magnetic tape). A minimum number of signals flow from the computer to the process. Therefore, the computer will normally have minimum control over the process. The data signals from the process are connected to the process input computer circuits and/or to the analog tape recorder. Thus, the data acquisition system monitors and records what goes on in the process without affecting the process functions in any manner.

When the analog tape is played back, the signals appear to the DACS computer exactly as real-time signals and can be handled by the computer in the same manner. By playing back the analog tape several times, a more selective retrieval of data is possible. If a DACS malfunction occurs in real-time, the data can still be retrieved by replaying the backup analog tape.

There are many reasons and applications for computer-assisted data acquisition. The following list is given to denote the almost universal applicability of data acquisition:

1. Record keeping: Many applications and industries must keep expensive records of processes. For example, airlines are required by law to record aircraft flight performance.
2. Telemetry is basically data acquisition and data logging from a remote or inaccessible process.
3. Data editing and data reduction applications involve processing large quantities of data which in many cases are redundant or non-significant. The computer program is called upon to reduce this data to manageable proportions.

4. Many processes have been designed on an empirical basis. Data logging and data acquisition is used to collect sufficient data to more fully understand the process. This data can be for process optimization and for model building.
5. By continuously monitoring parameters within a process, alarm conditions can be detected. Each value is continuously compared against preset conditions. Whenever a parameter is not within limits or the status does not check as expected, an appropriate alert can be given.
6. By carrying the limit violation check procedure further, trend predictions can be made. Thus, when rate of change of parameters exceeds prescribed limits, remedial action can be taken.
7. Important application areas are quality control and industrial testing. By using computer-aided test stands, it is possible to do 100% testing on production line components and produce printed records of the test results. For many military specification components, 100% testing is mandatory. Furthermore, the statistical data accumulated is necessary for various quality control work and product assurance. Computer-assisted testing makes the procedure economically practical. The degree of confidence that a customer has in a product is enhanced when he can obtain a printed record of the product's acceptance test results.

OPERATOR GUIDE/SUPERVISORY CONTROL

Almost all Operator Guide/Supervisory Control (OG/SC) systems embody data acquisition principles. In order to control a process, one must first measure and monitor it. The data acquisition feature is the monitor portion of OG/SC. Thus, data acquisition is the first step to control. When starting a control project, one must obtain sufficient data about the process through a planned data acquisition system. The data acquisition finally becomes the process I/O of the control system. OG/SC systems are characterized by a two-way data flow between the process and the computer. The computer monitors the process and based upon these measurements determines what control is necessary to make the system function in accordance with the program. To a first approximation, the difference between Operator Guide Control and Supervisory Control is one of how the computer interfaces with the process. Operator Guide Control usually implies the presence of a human operator who receives instructions from the control processor on an output display device, such as a typewriter. The operator makes adjustments and sets controls on set-point controllers based upon the control processor originated instructions. Supervisory control situations allow for direct control of the process by the computer via hardware electrical connection to the set-point controllers. The computer can make adjustments and sets control without operator intervention. Both categories rely upon direct computer data acquisition.

Most conventional processes rely upon analog controllers to hold various process variables within set points. Processes are usually made up of a number of cascaded operations with each step controlled by an analog controller. To adjust or optimize the process for particular modes of operation, each controller is set through its set-point control. The set-point may be reset during the process to compensate for various process perturbations.

When a computer controls the process, various modes of control are feasible. For example, in start-up control, the computer will first issue instructions to set the controllers to various set-points depending upon the process. The computer continually monitors the set-points on the controllers and the various controlled variables. Subsequent steps in the starting sequence are not permitted to commence until all values are within their prescribed tolerances. This type of control assures a reliable repeatable checkout and start-up procedure. Full documentation of all controls and variables are an implied by-product. Furthermore, by extending this concept, a standard operating procedure for the process is feasible.

Process optimization generally requires a mathematical model which has been defined on the basis of theory and/or data obtained using data logging techniques, etc. Successful optimization results in maximizing or minimizing parameters deemed important by management.

An attractive feature of the OG/SC type of computer control is that the process equipment is least disturbed. The only effect on the analog controllers is that they must be provided with remote set-point capability. Reversion to manual control is easily accomplished because the analog controllers are still in place. This provides an important backup capability for keeping the process running during a computer shutdown. In addition, the minor changes to the process instrumentation is an economic advantage.

Computer control can thus be installed in most existing processes without the necessity for building from the ground up.

DIRECT DIGITAL CONTROL

Direct Digital Control (DDC) is another advanced form of computer process control. In this mode of operation, the process will probably not run without the computer. The computer controls the process directly without the intervention of analog controllers. Computer-generated signals are used to control the process directly.

A DDC application generally requires re-instrumentation on existing processes. The proponents of DDC have advanced many arguments for this method of control. The marriage of the process to the computer provides many potential control techniques, not otherwise available, because of the tight control that is feasible.

A major advantage of DDC is the ease in which a parameter may be instrumented and controlled. A minor hardware addition in the process coupled with another program loop is all that is needed to control an additional variable. On the other hand, to add an analog controller is a major construction task compared with its DDC equivalent. To a large extent, the number of process loops that can be controlled is limited only by processor-controller and process I/O capabilities.

Analog controllers may be manually adjusted for best control for any given set of plant operating conditions. If these conditions change, analog controllers must be manually readjusted to retain optimum control. Readjustment is so time-consuming that it is usually impractical. In DDC, to change a critical time constant or to add a derivative function entails the addition of a few punched cards rather than major hardware. The effective characteristics of a controller are thus changed by inserting new cards into the deck. The flexibility and ease with which DDC can be altered is not possible with analog controllers.

Further Reading

For the benefit of inexperienced readers, the following IBM publications are recommended for further reading:

Principles of Data Acquisition Systems, Form E20-0090

Programming for Computer Control of a Cement Kiln, Form Z20-1753

High-Speed Data Acquisition in Low Energy Physics, Form E20-0171

The IBM 1800 Data Acquisition and Control System for Gas Turbine Engine Testing - Developmental, Form H20-0121

The IBM 1800 Data Acquisition and Control System for Gas Turbine Engine Testing - Production, Form H20-0123

Computer Control of the Continuous Hot-Dip Galvanizing Process, Form E20-0178

1800 Traffic Control System - Application Description, Form H20-0212

1800 Process Supervisory Program (PROSPRO/1800) - Application Description, Form H20-0261

TSX SAMPLE SYSTEM

The TSX Sample System was conceived and developed for two primary reasons. First, it demonstrates in a step-by-step fashion how to generate a TSX system from start to finish. Second, it illustrates some of the ways in which TSX can be used to actually control a process. The sample system is a working operating system which has proven itself in continuous service. For practical purposes, and in the interest of simplicity, an IBM 1800 Data Acquisition and Control System is linked to a Process Simulator to reproduce an actual continuous process which closely approximates a paper machine used in the manufacture of paper of diverse grades.

The general objectives of the system are to give better quality and quantity control. Its design basis included four specific criteria:

1. Provide periodic logs of all important variables within the system.
2. Establish closed-loop control over eight process variables.
3. Incorporate process operator control over forty process variables.
4. Display up-to-the minute management information about the process.

The sample system demonstrates how a number of functions may be achieved by using TSX, with relatively little effort on the user's part. The features of special significance in this example are

- the scheduling of periodic, semi-periodic, synchronous and asynchronous programs
- closed-loop control

- process job scheduling
- error recovery procedures
- system design to achieve maximum utilization of the time-sharing mode

SYSTEM DESIGN

The first thing that must be done in designing a control system is the definition of the process to be controlled; the second is the definition of the hardware (that is, the computer and its associated instrumentation); and thirdly, the design of the programming software. In this example, the process is referred to in terms of a paper machine, but those readers who are familiar with the paper industry will immediately realize that a paper manufacturing system in its full detail is quite complex, and for this reason, many of the paper machine control functions normally encountered are intentionally omitted. Among these are the handling of paper breaks and the use of instruments for on-line measurements of consistency, basis weight, and moisture content.

The paper process is designed to manufacture small amounts of paper of high quality grades. Orders arrive at the plant for varying quantities of paper of different paper grades. For example, orders may differ with respect to dimension, basis weight, strength properties, color, chemical additives, etc. The machine must be able under continuous operation to switch from one grade of paper (when the order quantity for that paper has been completed) to the next grade of paper. This means that grade change must be made as quickly and efficiently as possible to avoid undue and costly waste. Also, during the manufacture of a given grade of paper, very tight control over the process is necessary to ensure quality that is consistent with past orders for this paper. The control variables must be kept at setpoint, and any undue variation in other variables within the system must be recognized and the operator informed.

Input to the system must be provided for the updating of job files on the disk so that the computer knows continually what grades of paper are to be manufactured and in what sequence. Data on grade and quantity of paper arrives from the order department on data cards (see Figure 90). These cards are read and placed in files, on the disk, by the nonprocess time-shared portion of the operating system; the process portion then goes through this information sequentially.

Provision must be made for a process operator, through the medium of a console, to be able to obtain and/or update information about the entire process system at any time. This may include the changing of a setpoint on a setpoint station or to have instantly all available information on a given variable, such as its current value, its high and low limits, and conversion factors. The process operator

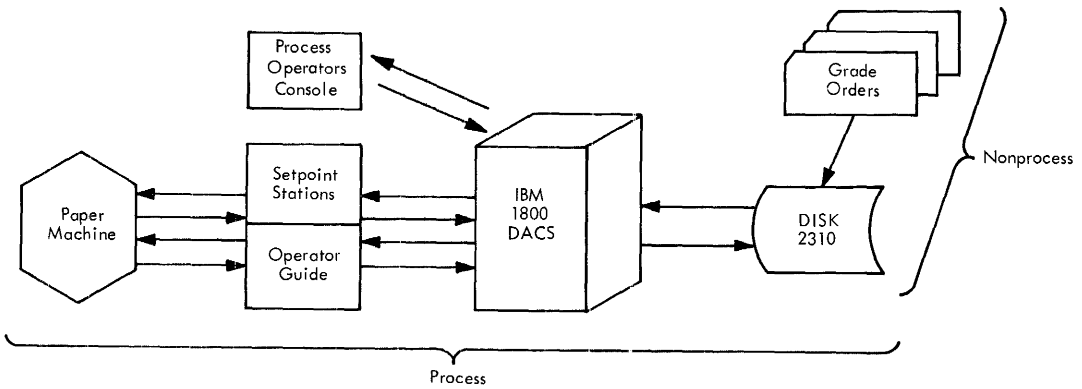


Figure 90. TSX Sample System Schematic Diagram

must also be able to modify the run-time of a particular grade of paper, and to change the sequence of process jobs that reside on the disk.

The 1800 hardware configuration for this system consists of the following:

- A two-microsecond 1801 Processor-Controller, with 32,768 words of core storage
- A 2310 Disk Storage unit, with two disk drives
- A 1443 Printer
- A 1442 Card Read/Punch Unit
- Three 1053 Printers

The 1443 Printer and 1442 Card Read/Punch, along with one of the disk drives are used for non-process work. This means that the card reader, the printer, and one disk drive constitute a basic stand-alone monitor system, while the process portion consists of the remaining disk drive, three 1053s, and process I/O.

PERIODIC PROGRAM SCHEDULER

Certain periodic functions must be handled within the system. First, every 20 seconds the closed-loop control program must be entered for a scan of the setpoint stations to see if they are on setpoint and, if they are not, to set up for the return of the setpoint positioners to their correct operating points. Second, every two minutes a scan of all operator guide points must be performed for limit checking, and any out-of-limit violations printed so that the process operator may bring these operating points within limits. Third, every quarter hour on the hour, a log of all variables within the system must be printed. Fourth, every hour, on the hour, a summary log of the previous hour's production must be printed for the attention of the process operator, the supervisor, and the foreman. Fifth, at the end of every shift (12:15 a.m., 8:15 a.m., and 4:15 p.m.) a shift-in log is printed which represents a summary of the past shift's production. Sixth, every Monday morning at 8:30, a weekly summary log is printed for close scrutiny by the supervisor and his foreman.

The first two of these functions are called periodic asynchronous functions because, while they are periodic in nature (that is, they occur every 20 seconds and every 2 minutes, respectively), they do not have to be in synchronization with real-time. The other functions are periodic synchronous since they are periodic and must be in synchronization with real-time, i.e., every hour, on the hour. To schedule the programs properly, a scheduler CALL COUNT subroutine is written in FORTRAN. The time-base method is used to determine when a periodic, synchronous program is next executed. In this way, a time is recorded for a given program that is chosen for execution. When the real-time clock is greater or equal to the time base specified, that program is scheduled for execution, i.e., it is queued, an end-time-sharing command is given, and the base-time is incremented by the period of the program.

SAMPLE SYSTEM ERROR DESIGN

Error procedures constitute one of the most important parts of any system. In the sample system, error procedures can be broken down into two phases, (1) checkpoint operation and, (2) system operation with certain I/O devices down or off-line.

Checkpoint Operation

The basic philosophy of checkpoint operation is that all system data needed by the computer to control the process must be stored in a location where it will not be destroyed. Thus, if normal operating data is destroyed, the data last saved can be retrieved and utilized. In the sample system, this is handled by having, on disk, two files that are used to record the process variables and certain conversion factors. One of these files is a static file, the other being dynamic in nature. Whenever a variable is modified in the system, one of the two files is updated with the new information. For example, if the operator at the process operator's console changes the setpoint of one of the setpoint stations, this information is recorded both in INSKEL COMMON (working data) and in a file on disk. If, at a later date, something should happen that would cause either a reload or a restart of the system, all operational data, such as setpoint values, limits on operator guide points, conversion factors, switches, can be read from disk to reinitialize INSKEL COMMON to the most updated valid values.

FILE1, the static file, contains all conversion factors used in the system. This file is normally read into core at cold start time to update INSKEL COMMON with the necessary matrices containing conversion factors for setpoint and operator guide analog inputs. The only time INSKEL COMMON and FILE1 are modified is when the calibration programs, which run in a time-sharing mode, are executed. At that time, the conversion factors of selected points are updated in INSKEL COMMON and the values are updated in FILE1.

The second file in the system for checkpoint operation, the dynamic file, is labelled FILE3 and contains all of the dynamic variables within the system. These comprise the job number, the day, the time when the present grade of paper is completed, the setpoint for all of the closed loops, and the limits for all points under operator guide control. This file is updated whenever one of the dynamic variables is modified, such as at the start of a new grade of paper or the entry of data into the system from the process operator's console.

The system contains both a restart core load and a reload core load. The restart core load is referenced by all but the C. E, interrupt mainline. Whenever EAC initiates a restart, this core load is brought into core storage. The system then reads the static and dynamic files from disk to initialize INSKEL COMMON, makes sure that both the scheduler and the end-of-grade call count subroutines are functioning correctly, and checks to see if an end-of-grade has occurred during the restart procedure. If so, it calls CHAIN to the mainline core load, GRADE. Otherwise, it calls VIAQ, and the system has restarted properly.

The reload core load is similar to the cold start core load, except there is a decision point within the core load to determine whether a reload or a cold start condition has occurred. Normally, sense switch 6 is turned on at cold start time. The cold start core load interrogates sense switch 6, and if the switch is on, the program goes through its normal cold start procedure. If sense switch 6 is off, the cold start routine assumes a reload has occurred and reads both static and dynamic files, starts the scheduler running again, and performs other functions that are similar in scope to those that occur in the restart core load. The normal cold start procedure, therefore, is to set sense switch 6 on and, when the system is up and running, turn that switch off, so that if a reload occurs, the system recovers correctly.

I/O Error Procedures

The system is designed so that it can function with only one of the two disk drives in operation. Thus, with a hardware failure on either disk drive, the system is able to run at full capacity on the process. This requires a two-disk system in which all process-oriented functions, such as cold start, the resident skeleton, system save areas, and all of the process core loads, are on logical drive 1, and all nonprocess functions, such as DUP, FORTRAN Compiler, Assembler, are on logical drive zero. Cold start is to logical drive 1; since all of the process is recorded on logical drive 1, there should never be a need to reference logical drive zero, unless time-sharing occurs.

In order to guarantee that the system does not initiate time-sharing unless an operator is present to specifically instruct it to do so, the cold start core load (COLDS) sets the CALL VIAQ time-sharing period to zero. This means that logical drive zero will be referenced only if the console interrupt button is depressed with sense switch 7 on. A cold start to logical drive 1 may, therefore, be performed without any fear of not having logical drive zero operational. Thus, if one of the two physical drives goes down, the process is still under control without any impairment to efficiency. When the drive is serviced, it can be restored on-line with the C. E. interrupt routine and time-sharing started. This allows for the C. E. to work on the drive while the system is still controlling the process. The system must also continue to operate if any two of the three 1053 Printers are down. This is achieved by setting the backup pattern for the 1053 Printers in cyclic order, such that

1053 Printer No. 2 backs-up 1053 Printer No. 1
1053 Printer No. 3 backs-up 1053 Printer No. 2
1053 Printer No. 1 backs-up 1053 Printer No. 3

This means that if one 1053 Printer goes down, it is backed-up, and if either of the other two printers goes down, the third printer backs-up the first two, so that as long as one 1053 Printer is operational, the system is still running.

Of the two other data processing I/O devices on the system, the 1443 Printer is used only on the time-sharing side of the system. If it therefore goes down, it will not affect the process side; in this event, there will be no printout of new job data loaded to the disk. The 1442 Card Read Punch is used by the process only at cold start time; if the system is up and running when the 1442 device goes down, the system is still able to control the process. The process can thus still be controlled by the 1800 hardware if, in the worst case, at cold start time, one of the two disk drives, two of the three 1053s, and the 1443 are all down.

To summarize, all that is needed for a cold start is a 1442 Card Read Punch, a 1053 Printer, and a 2310 Disk Storage Unit with two disk drives. After the system is placed on-line and becomes operational, only one 1053 Printer and one disk drive are required. The way the system is designed, it functions even if the final 1053 goes down, but with one restriction: no operator messages are printed. Closed-loop control is still in force, and the user is still able to enter information through the process operator's console, but he is not able to obtain information on the grade of paper being produced, and any out-of-limit violation that may occur.

The EAC printer for the system is defined as all three 1053s. This ensures that if an error condition does arise, the operator on at least one of the three 1053 printer stations is informed of the problem. To enable use of the C. E. interrupt routines (that form part of TSX) to place a disk drive, a 1443 Printer, and a 1053 Printer on-line or off-line, a C. E. core load is provided. This core load is called whenever it is necessary to use the C. E. interrupt routine. A special C. E. core load is needed within the system because the C. E. interrupt cannot be masked. This means that when the C. E. interrupt button is depressed, no matter what the masked status of the 1800 is, the interrupt will occur. It also signifies that the C. E. routine has no choice but to assume that the whole system is masked at the time of the interrupt, and it therefore returns the machine in this status. The interrupt mainline core load must therefore unmask back to the desired configuration at the time of the C. E. interrupt. In order to assure that this occurs, a special core load is called into core before the C. E. interrupt button is depressed. The core load types out a message indicating that the operator can now press the C. E. interrupt button; it then performs a pause. The C. E. interrupt causes the mainline program to drop through the pause. At this moment, the machine is masked back to the user's defined condition and a CALL VIAQ is performed.

Since the system has been designed for error recovery, restarts and reloads can be performed without loss of process control. While the system is on-line, the C. E. is able to carry out both preventive and corrective maintenance on the 1443 Printer and on two of the 1053 Printers. Once these devices become serviceable, they are restored to on-line duty by use of the TSX C. E. interrupt routine, and the system continues operation in the normal mode. The C. E. Aux Storage routines are also used while the system is on-line.

CLOSED LOOP CONTROL

It is desired to have the sample system perform closed loop control on eight setpoint stations. The scan requirement demands that these eight stations be interrogated every 20 seconds to ensure they are operating at the right setpoint value. A decision is now made whether the closed loop control program is to be a core load or an INSKEL interrupt subroutine. If the closed loop control program is to be a core load, it would require core exchanges every 20 seconds in order for it to operate as specified. This would be very inefficient in that the large number of core exchanges would cut down the useful amount of processor-controller time. Also, time-sharing periods would be of such small magnitudes that a nonprocess job would take a great amount of time to execute. For these reasons, the closed loop control program is written in FORTRAN as an INSKEL CALL LEVEL interrupt subroutine; that is, it resides in permanent core.

The setpoint station couples the 1800 hardware to an analog control loop; the station receives information from the computer in the form of a pulse train, and transmits information back to the computer in the form of an electrical signal so that this may be read by analog input. This signal indicates the position of the setpoint. The output from the setpoint station to the process can be used to operate a pneumatic control valve directly or through a pneumatic relay. The setpoint station has a scale range of 0 to 100, divided into 2, 000 increments. One pulse output to the setpoint station drives the setpoint positioner one increment. The pulses must be spaced at least 15 milliseconds apart in order to drive the positioner at an effective rate. The sample system is programmed to anticipate setpoint positioner drift. Thus, the control loop more closely resembles direct digital control (DDC) than normal setpoint station operation.

The closed loop control program is divided into two sections. The scan section scans all eight setpoint stations and computes the number of pulses needed for each one; the output section outputs the pulses. A CALL LEVEL (10) command is given to enter the program and, depending on a switch setting, one of the two sections is executed. Every 20 seconds the scheduler sets up an entry into the scan section. Once the scan section has completed its task, it initiates the output section.

The switch for the program is now set so that future entries will enter the output section of the closed loop program before another 20-second period has elapsed. This is necessary because the program must not stay on the interrupt level of the closed loop control program during the output of the pulse chain needed to move the setpoint station, which would tie up the system too long. For example, if 200 increments are put out to the setpoint station, the necessary loop would take 3 seconds for execution. Therefore, once the output section has outputted a pulse to the setpoint station, it calls for timer B to set up reentry to itself in 15 milliseconds. It then exits from the interrupt level so that computation or execution on the mainline level can continue. Fifteen milliseconds later, the timer runs out and the subroutine associated with the timer executes Call LEVEL back to the closed loop control program. At this point, if another pulse is needed for any of the setpoint stations, it is outputted and a Call TIMER is again given. If all of the desired pulses have been given, no Call TIMER is made and the program exits. The closed loop control program will not be entered again until 20 seconds have elapsed from the last entry to the scan section.

This method of control provides background and foreground operations with the TSX system, i.e., multi-programming. The closed loop control program constitutes the foreground job, and the program resident in variable core, the background job.

OPERATOR GUIDE CONTROL

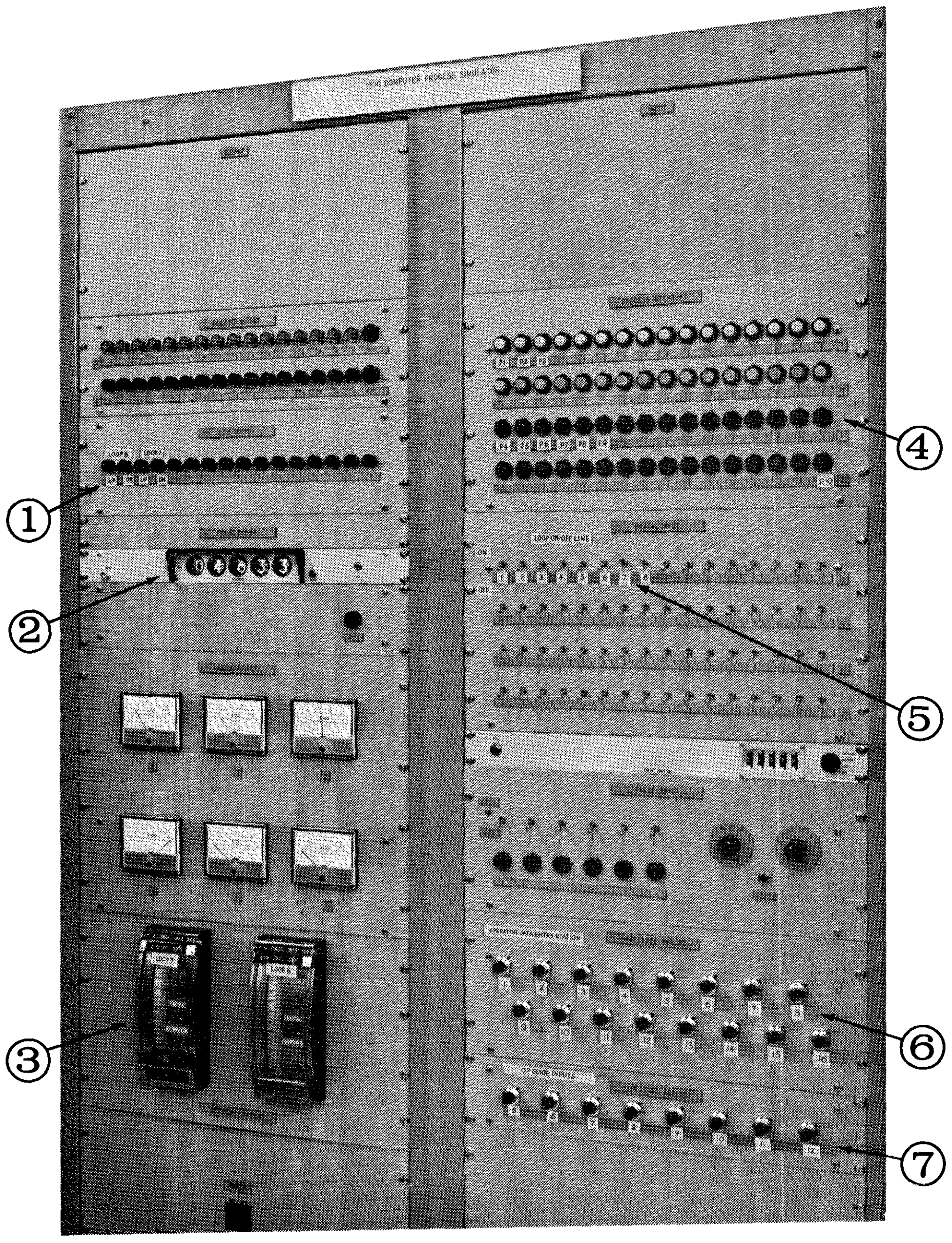
One of the system design criteria is to incorporate operator guide control over forty process variables. At the start of a grade, the operator is informed of the limits on the 40 variables he is responsible for controlling. Every two minutes these variables are scanned, and if an out-of-limit condition occurs on any of the variables, this information is printed for the operator. Also, the operator has the ability to perform a two-minute scan on demand so that he can quickly scan all points, when desired. At the same time, the operator can call for total information on any one point under control, i.e., its present value, its high and low limits, and conversion factors. The operator also has the option of changing the limits of a point or taking a point off operator guide control. To take a point off-line, he sets to their maximum and minimum values, all of the limits checked by the two-minute scan. In this manner, he is continuously informed of any erroneous out-of-limit condition.

SYSTEM DESIGN FOR OPTIMUM TIME-SHARING

Another desirable system requirement is to make optimum use of the nonprocess mode of operation on a time-shared basis. Among the activities that the user may wish to implement in a time-shared mode are instrument calibration, updating of process job files on disk, accounting, payroll, etc. As it is assumed that the user is going to make very heavy use of the time-shared mode, a 1443 Printer is included in the system. The system and list printers for the Nonprocess Monitor are both defined as the 1443 Printer so that all normal nonprocess output would be on that printer. With the design of the system such that only one disk is required for process work, logical drive zero can be switched among different nonprocess monitor disk cartridges. This allows each department within the company to have its own nonprocess monitor disk cartridge, with its own set of programs. When a department job is ready for processing, and computer time is available, all that is required to be done is to place the disk cartridge on disk drive zero and execute. With the 1442/1443 combination, the nonprocess user appears to have a small independent computer for his own, i.e., a computer with a card read punch, a printer, and one disk. A data processing computer is thus combined with a strictly control systems computer in one machine and under one operating system. In fact, if the user desired, he could well set up a closed shop, stack-job environment where nonprocess jobs submitted by the various departments within the company are run while the computer is controlling or monitoring the process.

PROCESS OPERATOR'S CONSOLE

One of the basic requirements of any process control system is a simple and efficient method to input data from a process operator. The operator must also be able to interrogate the process, with relative ease, to obtain any information required. There is no single standard approach to this problem, since the nature of the information needed by any one industry and, in fact, any plant within an industry, varies greatly. In the sample system, a portion of the 1800 Computer Process Simulator is used for the operator's console (see Figure 91). Also, one of the 1053s works in conjunction with the console. This means that the process operator has the console,



LEGEND

① Setpoint Station Movement Indicators	⑤ Digital Input Switches
② Digital Data Display (nixi tubes)	⑥ Operator Data Entry Dials
③ Setpoint Stations for Loops 7 & 8	⑦ Analog Inputs Under Operator Guide (8 of the 40 points)
④ Function Buttons	

Figure 91. 1800 Computer Process Simulator

along with a 1053, at his work station. Standard process I/O is used to operate the process operator's console; analog input is used to enter information from the operator's data entry dials; process interrupts are used as function buttons; digital input is used to define whether a certain closed loop is under control; digital output is used to indicate the direction of movement of a setpoint positioner when it goes through a control movement; and pulse output is used to count the number of increments given to all setpoint stations at any time.

Data Entry Dials

In normal practice the process plant operator has a definite need to enter digital data into the system. This may take the form of a new set of limits for an operator guide point, a new value for a setpoint station, or a new time for a particular grade of paper to have its production terminated. In order to accomplish this on the sample system, sixteen data entry dials are provided. Each data entry dial consists of a linear potentiometer connected to an analog input point. To obtain fine adjustment, a vernier scale is incorporated with each potentiometer. A major dial is graduated in scale from 0 to 10, while a minor dial (the short scale of the vernier) gives increments of subdivisions of each division recorded. Each data dial enters one digital number into the system. The process operator sets the potentiometer at one of the major divisions, i.e., 1, 2, . . . 9, and that digital integer is entered into the system. The digital integer entered is translated by the potentiometer to give out an analog value between 0 and -32, 000. This analog value is then converted by the data entry dial read routine to a floating point number between 0. 0 and 9. 9; this number is finally transformed and truncated to an integer value that lies within the range of 0 through 9. The data entry dial routine then transmits to the calling program a 16-element linear matrix containing the digital values for all 16 data entry dials. The calling routine can combine these integer values to produce the desired digit-coded information.

Function Buttons

Each function button is connected to one process interrupt. The operator depresses a function button depending upon the function required from the console. An example might be an operator call for demand scan of operator guide points; all that is required of the operator is to depress the proper button. This causes a process interrupt; the two minute normal scan program is then loaded to core and executed. Some of the process operator's console programs are also executed via the data entry dials. The operator sets the required data entry dials, and pushes the associated function button. The proper program is loaded; it interrogates the data dials; and then performs the operation requested.

Digital Input Switches

These 16 switches are connected to one digital input word. In the sample system, eight of these switches are used to define whether a given closed-loop is on-line or off-line. Before the system outputs a pulse to a setpoint station or scans the closed-loop control point, it first executes a read-and-expand function on this digital input word. It then interrogates each bit read to determine whether that loop is on-line or off-line. If the loop is off-line, no control function is performed.

Digital Data Display

The data output side of the process operator's console is divided into three sections. The first section is a set of nixi tubes for displaying digital data. These are used as a counter of the number of pulses given to the setpoint stations. The counter is manually reset from the operator's console. In a normal operating system, there is very little need for such a device, but it was found desirable to have it for experimental purposes on this system.

Setpoint Station Movement Indicators

This section consists of 16 lights, two for each setpoint station. One of the pair of lights per loop indicates whether the setpoint positioner is moving up, while the other light indicates the setpoint positioner is moving down. The lights are operated by digital output.

IBM 1053 Printer

This is used for outputting data, such as periodic logs, that the user requests. The process operator's console is simple to use and provides an efficient solution for man/machine interface problems on the system.

SYSTEM DOCUMENTATION

One of the most important and the most often neglected aspects of control systems is proper system documentation. The following paragraphs explain the need for this vital part of planning and installation.

Aid in System Debugging

A control system application is not solely a conglomeration of multiple programs working independently of one another. It is, rather, one big program that has been highly segmented. In order to insure correct communications between all program segments, a condensed picture of the system is needed, to show the interconnections between all program segments, their interactions, what common variables they reference, and how these programs are sequentially or randomly executed.

Basic System Documentation Needed

Documentation that can be maintained is the key to understanding and success of the system. The first supporting procedural device is an overall system flow chart that illustrates the basic interactions between programs (see Figure 92). The second is the maintenance of a list of all tables and variables that are referenced by more than one program (see Tables 14 and 15). Information is required to show what programs reference these system variables and which programs modify them. The third item is a specification sheet describing each log in the system (see Table 16). The fourth item is a specification sheet on each program, that briefly tells what the program does, what variables are referenced, what files are referenced, what programs call this program, what other programs this program calls, and any other information that has to do with the integration of this program into the system (see Table 17).

DESCRIPTION OF SAMPLE SYSTEM FLOWCHART

The sample system flowchart (see Figure 92) illustrates overall system operation and points out sequential, random, and time-based operations in a single flowchart. The flowchart is concerned only with the relationship between various programs within the system, not with the happenings within a program. It describes how programs are initiated and how they exit. Note that some program blocks are self-contained; that is, they neither cause another program to be initiated nor are they initiated by another program. They are triggered only by a random event, such as a process interrupt, and, once executed, perform only an exit. On the sample system flowchart, both chaining and queueing from one program to another are represented in the same manner because they constitute, basically, the same function. The main difference between them is that a CALL QUEUE may allow other programs to "sneak in" between the sequential execution of the calling program and the called program. Note also that two programs, SCHED and TCONT, call themselves with a time delay and thus insure that they are periodically executed. The TCONT program may have its chain of execution broken by the program TEBRT. This means that TCONT is periodic only for a certain duration of time and that the operator may control that duration of time if he so desires (see Table 17: Program Data Sheets).

CODING TECHNIQUES

The sample system depicts the employment of several coding techniques which may be of value to the user. Three techniques are discussed in this section. The first two are examples of good programming coding practice and should help to eliminate the type of system problems a user may encounter at TSX system installation time. The third coding technique illustrates how one of the functions on analog input can be used to advantage.

Use of INSKEL COMMON

INSKEL COMMON is one of the principal means by which coreloads and subroutines within the system can communicate with one another. This unique labelled common area is mapped each time a new core load is built. This means that problems can arise if two different core loads have INSKEL COMMON mapped differently. In order to avoid such conflicts in its use, all core loads must map INSKEL COMMON in the same manner. Also, for good coding practice, they should use the same variable names for every variable in INSKEL COMMON. The only way to insure this common usage is for the user to keypunch only one COMMON card which may subsequently be duplicated and inserted into each core load source deck. If at some later time a change is made to INSKEL COMMON, the new COMMON card may be keypunched, copies made, and all old COMMON cards in source decks replaced. In this way, the user ensures that INSKEL COMMON is defined identically for all core loads.

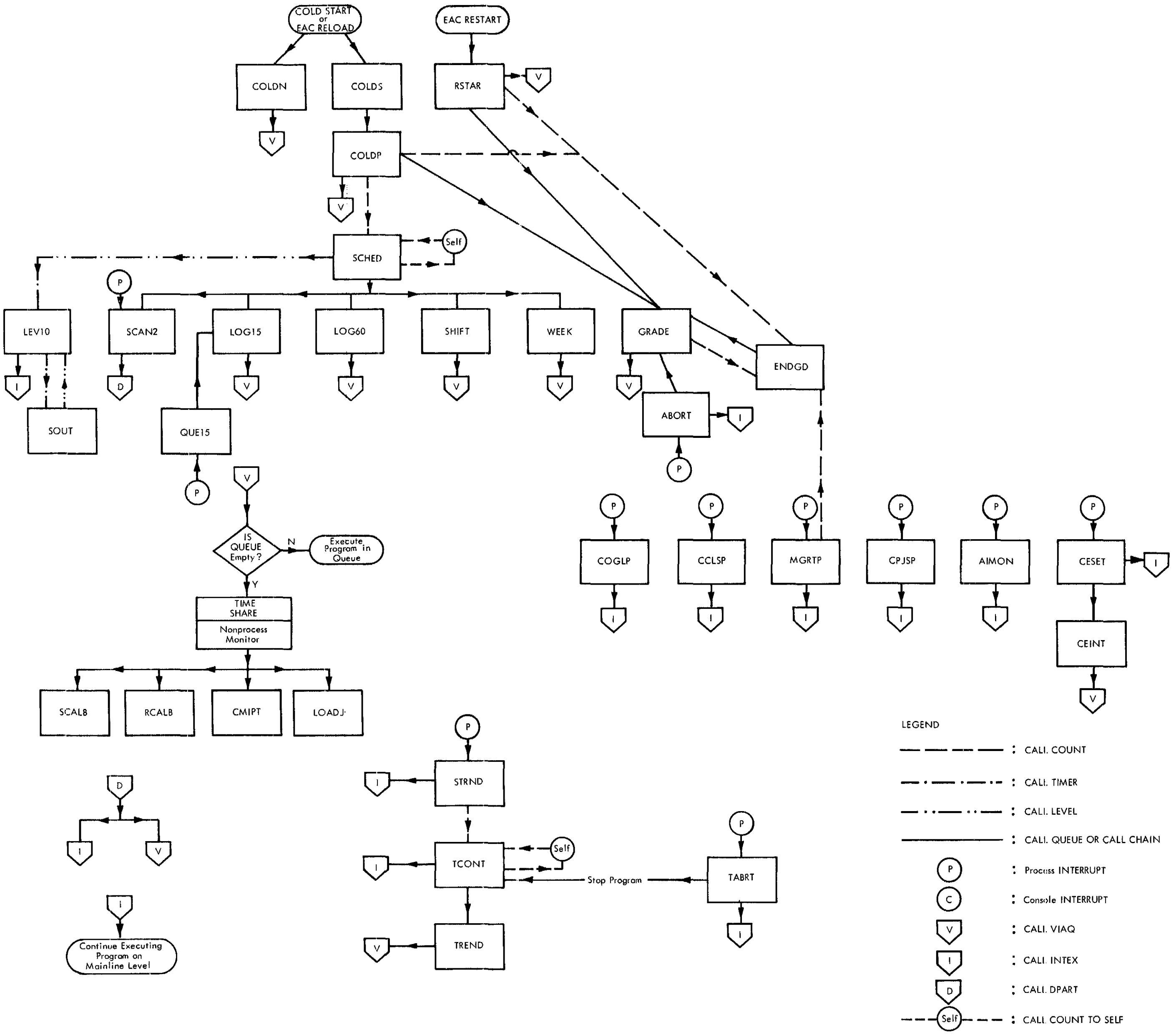


Figure 92. TSX Sample System Flow Chart

Table 14. TSX Sample System Table of Variables

System Variable	Used by	Function
SW0	SCHED, SOUT, RSTAR, COLDP, GRADE	Close Loop Control Stop
SW1	SCHED, COLDP	Update day-of-week switch
SW2	SCHED	Output Monday morning log switch
SW3	SCHED, ENDGD, RSTAR, COLDP, GRADE, MGRTP, CPJSP	Process Control Stop Switch
SW4	SCHED, LEV10, SOUT	LEV10 program section switch
SW5	SCHED, COLDP	SCAN2 Execution switch
DAY	SCHED, RSTAR, COLDP, GRADE, SCAN2, LOG15, LOG60, SHIFT, COGLP, CCLSP, MGRTP, CPJSP, AIMON	Day of week
JOBN	RSTAR, COLDP, GRADE, MGRTP, CPJSP, SPECL	Next process job
VALUE	LEV10	LEV10 Data input area
RANGE	COLDN, RSTAR, COLDP, GRADE, LOG15, TREND, CCLSP, AIMON, SPECL, SCALB, RCALB	Range of analog input values for setpoint stations
LOW	COLDN, RSTAR, COLDP, GRADE, LOG15, TREND, CCLSP, AIMON, SPECL, SCALB, RCALB	Low analog input values of setpoint stations
SETPT	LEV10, RSTAR, COLDP, GRADE, CCLSP, AIMON, SPECL	Operating points for setpoint stations
COUMT	LEV10	Number of pulses to be given to setpoint stations
OFFLN	LEV10	Off-line indicator for setpoint stations
AHL	RSTAR, COLDP, GRADE, LIMIT, COGLP, SPECL	High limits for op-guide points
ALL	RSTAR, COLDP, GRADE, LIMIT, COGLP, SPECL	Low limits for op-guide points
A	COLDN, RSTAR, COLDP, LOG15, TREND, AIMON, SPECL, SCALB, RCALB	Conversion factor A for op-guide points
B	COLDN, RSTAR, COLDP, LOG15, TREND, AIMON, SPECL, SCALB, RCALB	Conversion factor B for op-guide points
IBASE	SCHED, COLDP	Base time for LOG15
IBASZ	SCHED, COLDP	Base time for LOG60
IBAZZ	SCHED, COLDP	Base time for SHIFT
G	CONVR, COLDN, COLDP, CMIPT	Conversion factor G for data entry dials
H	CONVR, COLDN, COLDP, CMIPT	Conversion factor H for data entry dials
IENDT	RSTAR, COLDP, GRADE, MGRTP, CPJSP	Grade termination time
IPERD	STRND, TCONT	TREND execution period
ITCNT	TCONT, TABRT, STRND	Number of times TREND is to be executed
IPONT	TREND, STRND	Point or Loop for TREND to log

Table 15. Disk File Organization

FILE	RECORD	VARIABLES	REFERENCING PROGRAMS
1	1	RANGE, LOW, A, B	COLDN, RSTAR, COLDP, SPECL, SCALB, RCALB
1	2	G, H	COLDN, COLDP
2	1-100	I, ITIME, SETPT, AHL, ALL	GRADE, SPECL, CMIPT
3	1	JOBN, DAY, IENDT SW3	RSTAR, COLDP, GRADE, MGRTP, CPJSP
3	2	SETPT	RSTAR, COLDP, GRADE, CCLSP
3	3	AHL, ALL	RSTAR, COLDP, GRADE, COGLP

Table 16. Log Description

NAME	TYPE	INFORMATION	RECIPIENTS
SCANZ	Periodic/Asynchronous (2 minute period)	Out of Limit Conditions on operator guide points	Process Operator
LOG15	Periodic/Synchronous (15 minute period on the quarter hour)	Present value of all analog input points	Process Operator, Foreman
LOG60	Periodic/Synchronous (1 hour period on the hour)	Summary of last hour's production	Process Operator, Foreman, Supervisor
SHIFT	Periodic/Synchronous (8 hour period at 0:15, 8:15, and 16.15)	Summary of last shift's production	Process Operator, Foreman, Supervisor
WEEK	Periodic/Synchronous (every week at 8:30 Monday morning)	Summary of last week's production	Foreman, Supervisor

Use of FORTRAN Files

FORTRAN files form the second means of communications among programs in TSX. The use of FORTRAN files, when compared to INSKEL COMMON, introduces an added complication. Not only are the files mapped at core load build time according to the number of records and the size of a record, but, in addition, each time they are read or written, the definition of the file contents takes place. At read/write time, the relative locations of variables within the record are mapped. In order, therefore, to insure proper communications between all programs that use FORTRAN files, the user must insure that all defined file cards, asterisk file cards, and read/write statements to files and records are the same. This may be accomplished by punching only one card for each of the above file reference statements and duplicating this card where needed in source programs. In this way, the user insures that the variable list is the same for each record within the system.

Use of Analog Input Read-and-Expand Function

The read-and-expand function for the AISQN subroutine is the most versatile of all of the analog input calls. With this call the user is able to handle each analog input point as it comes in, and to simultaneously insure that the rate of input of analog input points is a maximum. The read-and-transfer function operates as follows: after one analog input point has been read into core, and conversion has been started on the next sequential point, the analog input subroutine transfers control to a user's subroutine, which can manipulate this point in any desired way while the next analog input point is being converted. The value just read can be transformed into engineering units, checked for out-of-limit conditions, and used in setting-up a control function.

All of these manipulations can be performed within the user's subroutine. This allows the system to achieve maximum overlap of data input and computation. In the sample system, the read-and-transfer function is used only in the SCAN2 program; it could also have been effectively used in the LEV10 program and, in fact, wherever analog input is used.

SYSTEM GENERATION

The entire system generation procedure for the sample system can be viewed on a step-by-step basis from the system and list printer output listings (see Program Listings No. 9: System Generation). These cover the initial writing of addresses on the disk cartridges, to the calibration of instruments (before a cold start is performed). Two items are excluded from the listings: the assembly of TASK and the assembly of the System Director. These two source decks have already been assembled and are in available object format.

Step 1. The first step in generating the system is the writing of addresses on the disk. Step 1, in the listing, shows the initialization of disks on disk drives 0 and 1. Note that at the time addresses were written on drive 0, there were no defective cylinders. When addresses were written on drive 1, three cylinders proved defective. The operator message states that logical cylinders /0013 are defective, indicating that there are three consecutive defective cylinders. Logical cylinder /0012 is operational, but there is a three-cylinder gap before logical cylinder /0013 is reached. Except for one restriction on its use, this disk is acceptable for TSX system operation. The restriction (given by an error message) is that a skeleton must not be built using this storage device; that is, the disk cartridge cannot be used for logical drive zero for an on-line system.

Step 2. The IBM Nonprocess System is loaded to the disk via the System Loader. The first item printed out is the Assignment Table, which lists all hardware devices defined for this configuration, the interrupt level of each device, and the bit position within the Interrupt Level Status Word (ILSW) where the interrupt bit for each device is located. After the Assignment Table is built, absolute programs are loaded to disk. These are followed by IBM-supplied subroutines.

The final item in the system load operation is the *DEDIT function which instructs the system to reserve 20 disk cylinders for the buffering of messages and that the object machine will contain 32K words of core storage.

Steps 3 through 8. These steps are concerned with the organization of the disks for this particular installation. Step 3 defines the sample system as a two-drive system; drives 0 and 1 are then labelled. Steps 6 and 7 define the configuration of these two drives. A differentiation is now made between the process and nonprocess drives. All control programs and components associated with the process, such as the skeleton and EAC, are located on logical drive 1 while all nonprocess functions such as the FORTRAN Compiler and the Assembler are located on logical drive 0. This is the initial stage in the organization of the sample system. Step 8 reserves disk space for data files used by the system; the values that will be stored in the files are not defined at this time.

Step 9. In step 9, the System Director, in object format, is stored from cards to a relocatable area on logical drive 1.

Steps 10 through 23. These steps set out the compilation and assembly of all system subroutines. These include INSKEL interrupt subroutines, COUNT subroutines, a solitary TIMER subroutine, and normally-called subroutines. As these are compiled or assembled, they are stored on disk in relocatable format so that they will be readily available at skeleton build time. Step 23 is a checkpoint along the system generation route; the *DUMPLET verifies that all subroutines necessary for the creation of the skeleton are properly stored on disk.

Step 24. The actual System Skeleton building phase is now carried out. TASK is reloaded to core from cards for the first time since the commencement of system generation. Note that the // JOB card for the skeleton build function contains an "A" in column 15, which defines a two-drive system; this card must be identical to the // JOB card which precedes the final // END OF ALL JOBS.

At the commencement of skeleton build, a KOC error message is printed, signifying that the ANINT subroutine (an I/O subroutine) has missed an interrupt branch table entry. The error message states that IAC code /0023 is missing from the Master Interrupt Branch Table. Since this EAC code concerns the comparator feature on the Analog Input Expander, and there is no comparator on the sample system, the error code can be ignored. After the skeleton map is printed, a series of K13 error messages is printed, indicating that certain named programs have not yet been built. Since these programs cannot be built until the skeleton itself has been built, these errors can be ignored. Note that a K13 error would be significant if an operator performed a skeleton rebuild and obtained a K13 error message for a program that should have been on disk. Once the skeleton is built, it is time to compile and build the mainline core loads.

Steps 25 through 45. All system process core loads are now built and stored in core image format on disk. At this point in time, TASK is still resident in core controlling the Nonprocess Monitor. Step 30 is of special interest; three dummy core loads are provided in the system to be referenced on a DUP *DELETE card, when it is desired to modify any of the core loads in the system. For example, if core load GRADE contains an error, and it is desirable to recompile GRADE and restore it on disk, the first thing that must be carried out before this can be done is to delete the old version of GRADE. If, however, the system is on-line at the time GRADE is deleted, a core load must be available on disk which can be loaded and executed when the core load name GRADE is called after its deletion, but before storage of its replacement. Since a straight delete of GRADE is not possible, it is preferable to replace GRADE with a dummy core load until such time as its new version becomes resident on disk.

Between steps 44 and 45, TASK is eliminated from core, and system operation is initiated by a cold start which specifies COLDN as the initial core load. The function of COLDN is to provide perpetual time-sharing so that the Nonprocess Monitor may be used to execute user-written nonprocess core loads during system generation. It is necessary to execute these nonprocess core loads during system generation because there are certain data areas that must be initialized prior to a process cold start. There are also four nonprocess core loads that will be executed in the time-shared mode, at a later time, that affect process operation.

Step 45 is concerned with the compilation and execution of a one-time-only core load, named SPECL, which is used to place initial values in the process.

Steps 46 through 51. Once the job file is initialized, step 46 compiles and stores the first of the calibration core loads. As soon as this core load is stored on disk, it is executed at step 47 to obtain initial calibration factors for closed loops 7 and 8. Steps 48 through 51 cover the calibration of other analog input points in the system.

Steps 52 through 54. At step 52, the program which will be used to update the job files on disk, as new grades of paper are placed in the process job schedule, is compiled and stored. It is executed at step 53 to ensure that it functions correctly. Step 54 is the final step in the system generation procedure; an *DUMPLET is performed on both drives to ensure that all core loads and data areas are correctly defined on disk. At this point, the sample system is ready to be placed on-line to control the paper machine.

ON-LINE OUTPUT FROM THE SAMPLE SYSTEM

Samples of output listings obtained during on-line continuous operation of the sample system are given at the end of this section (see Program Listing No. 10: On-Line Process Output). The specimen listings illustrate process operator, supervisor, and time-shared output.

Output 1. This is generated at cold start time on the process operator's 1053 Printer. It reveals the cold start procedure and continues with a typical sample output over the next few hours of processing on the same printer. Note that at cold start time, a process job sequence number of zero is supplied to the system; the sample system went, therefore, into a production stop state. Two minutes later, the first process job number is entered into the system from the process operator's console. When the process operator receives a signal that the paper machine is ready to go on-line with this grade of paper, he depresses the abort-grade button which initiates the production of grade 12345; this grade had been entered into the queue sequence a minute earlier. At the start of grade 12345, the production time is printed as one hour and 29 minutes; the start time is recorded as 8:03 a.m., Friday. At this stage, the high and low limits for all operator guide points as well as set point values for the set point stations are printed. At 8:05 a.m., the operator at the process operator's console changes the set point value for loop 7 to a value of 50. Note that the normal scan of the operator guide points is executed every two minutes. At 8:15 a.m., the first quarter hour log is printed. As can be seen from the listing, closed loop 7 is now on set point value. It should be noted that the set point range for acceptable values is within plus or minus one of the actual set point value. Following the 8:15 a.m. quarter hour log, the shift-end log is printed. At 9:00 a.m., the first of the periodic one-hour logs is printed.

Output 2. This output is on the process operator's 1053 printer and shows three of the operator console functions being executed. The first function occurs at 11:28 a.m., Friday and consists of changing the high and low limits on operator guide point 5. Note that two minutes later point 5 is recorded as being out of limits, and calls for a demand scan, at which time no limit violation is recorded. Four minutes later, at 11:34 a.m., the operator calls for complete information on operator-guide point 5. The value of point 5 is recorded; its high and low limits and the two conversion factors are printed.

Output 3. Output 3 shows two of the other functions which can be performed from the process operator's console. The first is the calling of a trend log on operator guide point 6. The operator specifies to the system through the console that he wants a trend log on operator guide point 6 every ten seconds, and that he wants this trend log repeated 300 times. This is followed by the output from the trend log.

Between the normal scan at 9:17 a.m. and the next normal scan at 9:19 a.m., eleven values of the trend log are printed. (In normal operations, within a two-minute period, twelve trend logs would be printed.) The reason for the eleven printed values is because the last trend log and the normal scan are both queued at the same time, and that the normal scan is queued with a higher value than the trend log. The normal scan is therefore printed first with only eleven trend logs between it and the preceding normal scan. After only nineteen trend logs have been printed, the operator decides to abort the trend operation before it runs to completion, by depressing the abort trend log function button on his console. At 9:21 a.m., the sequence of process jobs is changed when the operator enters a new sequence number of zero into the system. This is done because the operator wants the computer to go into a suspended state, as far as the paper machine is concerned, when the current job runs to completion. It also means that the paper machine is being taken off computer control. At 9:23 a.m., the manufacturing process of this grade of paper comes to an end; the system then recognizes that the operator has changed job sequences to indicate that it is now time for a production stop. This occurs at 9:25 a.m. The operator now enters a new job number, 35, so that when the paper machine is ready to commence a new cycle of operation, all he need do is to hit the grade abort function switch to bring the process back under control.

Output 4. Output 4 shows how the operator changes the job sequence so that a different grade of paper can be made when a present grade under production is completed. At 12:26, the production on the preceding grade of paper being completed, the new grade, 4530 (which the operator has entered into the system), is initiated.

Output 5. Output 5 shows the printout on the process operator's 1053 Printer at the time of a reload condition. At this point, an intentional reload of the system is caused by bringing about an op-code violation in the Skeleton by depressing stop, reset, start. Following the normal scan at 9:06 a.m., the EAC reload message is printed. The cold start core load which determines that this is a reload operation then prints the message: Process Restart Checkpoint. At this time, all of the variables in INSKEL COMMON have been reinitialized, and the system is back in control of the process. Following the normal scan at 9:07 a.m., the quarter hour log, the one hour log and the shift-end log are printed. These are printed at this time to assure the operator that the system is running correctly and that all functions are operating.

Note that at 9:10 a.m. the operator performs a demand scan and is told that a limit violation has occurred on point 5.

Output 6. Output 6 demonstrates what happens when an error occurs in a nonprocess time-shared job. On Wednesday at 13:56, a time-shared nonprocess program is executed while the process is being monitored. This time-shared program which contains a FORTRAN I/O error thus causes the system to abort that job. Note that this has no effect on the control of the process.

Output 7. Output 7 demonstrates the use of the TSX C.E. interrupt routine to bring logical drive 0 on-line. (Logical drive 0 had been taken off-line prior to 9:02 a.m. on Tuesday.) This means that the system is operating with only the process cartridge on-line. At 9:02 a.m., it was decided to run some non-process jobs; it therefore became necessary to place a nonprocess cartridge on logical drive zero and to bring that drive on-line.

Before the drive is brought on-line through the use of the TSX C.E. interrupt program, the operator first calls in his C.E. unmasked core load via the console interrupt. Once the message is printed, stating that this core load is in core, he depresses the C. E. interrupt button. This brings the C. E. interrupt routine into core; the operator then proceeds to bring logical drive 0 on-line. Once the drive is on-line, normal process control continues, and the operator is now able to execute time-sharing jobs.

Output 8. Output 8 is on the foremen's 1053 Printer. This shows two things: first, that information is given to the foreman, and the extent to which it varies from that supplied to the process operator. It also shows at what time system operation is terminated after a period of continuous operation. Note too, that the Monday morning report is printed exactly on time at 8:30 a.m.

Output 9. Output 9 is from the nonprocess time-shared side of the sample system. It shows what is printed on the 1443 Printer at the time the job files on disk are updated with data on new grades of paper to be produced.

Table 17. Program Data Sheets

These are specifications sheets of each program used in the TSX Sample System (see Basic System Documentation Needed). Each encircled listing number in the table corresponds to its exact counterpart in Program Listing No. 9: System Generation.

Listing No. 10

Program. SCHED

Type. INSKEL CALL COUNT Subroutine

Description. This subroutine schedules the periodic execution of the programs listed in the external statement. This subroutine is entered every 20 seconds.

System Variables Referenced. SW0, SW1, SW2, SW3, SW4, SW5, DAY, IBASE, IBASZ, IBAZZ.

Files Referenced. None

Programs Called. SCHED, SCAN2, LOG15, LOG60, SHIFT, WEEK

Calling Programs. RSTAR, COLDP, GRADE

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 12

Program. SOUT

Type. INSKEL CALL TIMER Subroutine

Description. This subroutine services timer 5 and is used for initiating entries into the level 10 subroutine for the outputting of pulses to the set point stations.

System Variables Referenced. SW0, SW4

Files Referenced. None

Programs Called. None

Calling Programs. LEV10

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 11

Program. LEV10

Type. INSKEL INTERRUPT Subroutine

Description. This subroutine does closed loop control of eight set point stations. Every twenty seconds, the scheduler subroutine gives a call level to this routine and sets SW4 to point to the scan section such that all eight points are scanned. When the scan is finished, the output to each on-line station is computed and the first pulse output is given. Timer B is then used to set-up fifteen millisecond entries into the subroutine so that all necessary pulses are given.

System Variables Referenced. OFFLN (1-8), SW4, VALUE (6-13, 15) SETPT (1-8), COUMT (1-8)

Files Referenced. None

Programs Called. SOUT

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 13

Program. QUE15

Type. INSKEL INTERRUPT Subroutine

Description. This subroutine queues the fifteen minute log routine on demand.

System Variables Referenced. None

Files Referenced. None

Programs Called. LOG15

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

14

Listing No.

Program. TCONT

Type. INSKEL CALL COUNT Subroutine

Description. This subroutine periodically queues the trend log program the number of times specified.

System Variables Referenced. ITCNT

Files Referenced. None

Programs Called. TREND, TCONT

Calling Programs. TCONT

System Subroutines Called. None

Restart Core Load. N/A

15

Listing No.

Program. TABRT

Type. INSKEL INTERRUPT Subroutine

Description. This subroutine aborts the trend log on demand.

System Variables Referenced. ITCNT

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

16

Listing No.

Program. GETVL

Type. USER Subroutine

Description. This subroutine reads the analog input value for each of the sixteen data entry dials.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. CONVR, CMIPT

System Subroutines Called. None

Restart Core Load. N/A

17

Listing No.

Program. CONVR

Type. USER Subroutine

Description. This subroutine scans the 16 data entry dials at the process operator's console and converts them to an integer value with range 0 to 9.

System Variables Referenced. G(1-16), H(1-16)

Files Referenced. None

Programs Called. None

Calling Programs. COGLP, CLLSP, MGRTP, CPJSP, STRND, AIMON

Systems Subroutines Called. GETVL

Restart Core Load. N/A

18

Listing No.

Program. PTIME

Type. USER Subroutine

Description. This subroutine reads the clock and converts the time to a floating point number with the decimal point separating hours and minutes.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. RSTAR, GRADE, SCAN2, LOG15, LOG60, SHIFT, WEEK, COGLP, CCLSP, CPJSP, AIMON

System Subroutines Called. None

Restart Core Load. N/A

19

Listing No.

Program. ISBAD

Type. USER Subroutine

Description. This subroutine gets the address of the entry point to a subroutine

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

19

Listing No.

Program. IADDR

Type. USER Subroutine

Description. This subroutine gets the address of a FORTRAN variable.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

20

Listing No.

Program. CESET

Type. INSKEL INTERRUPT Subroutine

Description. This subroutine queues the CE unmask program so that devices may be taken off-line or put on line.

System Variables Referenced. None

Files Referenced. None

Programs Called. CEINT

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 21

Program. ABORT

Type. INSKEL INTERRUPT Subroutine

Description. This subroutine queues the grade change program causing the present grade to be aborted.

System Variables Referenced. None

Files Referenced. None

Programs Called. GRADE

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 22

Program. ENDGD

Type. INSKEL CALL COUNT Subroutine

Description. This subroutine aborts the grade in progress when the run time for that grade has elapsed.

System Variables Referenced. SW3

Files Referenced. None

Programs Called. GRADE

Calling Programs. RSTAR, COLDP

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 25

Program. COLDN

Type. MAINLINE CORE LOAD

Description. This cold start core load is used to give perpetual time sharing.

System Variables Referenced. RANGE, LOW, A, B, G, H

Files Referenced.
File 1, Records 1 + 2

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. COLDN

Listing No. 26

Program. COLDS

Type. MAINLINE CORE LOAD

Description. This is the normal cold start core load. It sets time sharing time to zero so that console interrupt must have been pushed before logical drive zero is ever referenced. This core load chains to COLDP to actually cold start the process.

System Variables Referenced. None

Files Referenced. None

Programs Called. COLDP

Calling Programs. None

System Subroutines Called. None

Restart Core Load. COLDS

27

Listing No.

Program. RSTAR

Type. MAINLINE CORE LOAD

Description. This is the system restart core load. Whenever a restart condition occurs, this routine is loaded to variable core to make sure system constants in INSKEL COMMON are valid.

System Variables Referenced. SW0, SW3, RANGE, LOW, A, B, JOBN, DAY, IENDT, AHL, ALL, SETPT

Files Referenced.
File 1 Record 1
File 3 Records 1, 2, 3

Programs Called. SCHED, GRADE, ENDGD

Calling Programs. None

System Subroutines Called. PTIME

Restart Core Load. COLDS

28

Listing No.

Program. COLDP

Type. MAINLINE CORE LOAD

Description. This is the system process cold start and reload core load. If sense switch 6 is on, it does a process cold start and if sense switch 6 is off it assumes a reload condition has occurred so that it initializes the system to the last check point.

System Variables Referenced. SW0, SW1, SW3, SW5, RANGE, LOW, DAY JOBN, A, B, SETPT, AHL, ALL, IENDT, G, H

Files Referenced.
File 1 Records 1, 2
File 3 Records 1, 2, 3

Programs Called. SCHED, GRADE, ENDGD

Calling Programs. COLDS

System Subroutines Called. None

Restart Core Load. COLDS

29

Listing No.

Program. CEINT

Type. MAINLINE CORE LOAD

Description. This core load is for use with the CE interrupt. It makes sure that all levels are unmasked after use of the CE interrupt routine.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. CESET

System Subroutines Called. None

Restart Core Load. CEINT

30

Listing No.

Program. DUM

Type. MAINLINE CORE LOAD

Description. This is a dummy core load for use in replacing or deleting system core loads.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. DUM

30

Listing No.

Program. IDUM

Type. INTERRUPT CORE LOAD

Description. This is a dummy core load for use in replacing or deleting system core loads.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

31

Listing No.

Program. GRADE

Type. MAINLINE CORE LOAD

Description. This program starts the production of a new grade.

System Variables Referenced. SW0, SW3, DAY, JOBN, RANGE, LOW AHL, ALL, SETPT, IENDT

Files Referenced.
File 2
File 3, Records 1, 2, 3

Programs Called. SCHED

Calling Programs. ABORT, ENDGD, RSTAR, COLDP

System Subroutines Called. PTIME

Restart Core Load. RSTAR

30

Listing No.

Program. CDUM

Type. COMBINATION CORE LOAD

Description. This is a dummy core load for use in replacing or deleting system core loads.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. CDUM

32

Listing No.

Program. SCAN2

Type. COMBINATION CORE LOAD

Description. This combination core load scans all of the Op-guide points on the system and notes any limit violation to the operator.

System Variables Referenced. DAY

Files Referenced. None

Programs Called. None

Calling Programs. SCHED

System Subroutines Called. PTIME, LIMIT

Restart Core Load. RSTAR

32

Listing No.

Program. LIMIT

Type. USER Subroutine

Description. This subroutine is the subroutine to be used in the AIS read and transfer function of the mainline core load.

System Variables Referenced. AHL, ALL

Files Referenced. None

Programs Called. None

Calling Programs. SCAN2

System Subroutines Called. NONE

Restart Core Load. N/A

33

Listing No.

Program. LOG15

Type. MAINLINE CORE LOAD

Description. This is the fifteen minute log program which logs the values of all process variables in the system.

System Variables Referenced. DAY, B(1-40), A(1-40), LOW(1-8), RANGE(1-8)

Files Referenced. None

Programs Called. None

Calling Programs. SCHED, QUE15

System Subroutines Called. PTIME

Restart Core Load. RSTAR

34

Listing No.

Program. LOG60

Type. MAINLINE CORE LOAD

Description. This program puts out the hour log.

System Variables Referenced. DAY

Files Referenced. None

Programs Called. None

Calling Programs. SCHED

System Subroutines Called. PTIME

Restart Core Load. RSTAR

35

Listing No.

Program. SHIFT

Type. MAINLINE CORE LOAD

Description. This program outputs the shift log.

System Variables Referenced. DAY

Files Referenced. None

Programs Called. None

Calling Programs. SCHED

System Subroutines Called. PTIME

Restart Core Load. RSTAR

36

Listing No.

Program. WEEK

Type. MAINLINE CORE LOAD

Description. This program outputs the weekly Monday morning log.

System Variables Referenced. DAY

Files Referenced. NONE

Programs Called. None

Calling Programs. SCHED

System Subroutines Called. PTIME

Restart Core Load. RSTAR

38

Listing No.

Program. COGLP

Type. INTERRUPT CORE LOAD

Description. This core load changes the limits on operator-guide points at operator request.

System Variables Referenced. AHL, ALL, DAY

Files Referenced.
File 3,Record 3

Programs Called. None

Calling Programs. None

System Subroutines Called. PTIME, CONVR

Restart Core Load. N/A

37

Listing No.

Program. TREND

Type. MAINLINE CORE LOAD

Description. This is the trend log core load. It reads the value that the operator has asked. It is queued periodically by the TCONT subroutine with the period specified by the operator.

System Variables Referenced. IPONT, LOW, RANGE, A, B

Files Referenced. None

Programs Called. None

Calling Programs. TCONT

System Subroutines Called. None

Restart Core Load. RSTAR

39

Listing No.

Program. CCLSP

Type. INTERRUPT CORE LOAD

Description. This core load changes the set point value for a set point station upon operator request.

System Variables Referenced. RANGE, LOW, SETPT, DAY

Files Referenced.
File 3,Record 2

Programs Called. None

Calling Programs. None

System Subroutines Called. CONVR, PTIME

Restart Core Load. N/A

40

Listing No.

Program. MGRTP

Type. INTERRUPT CORE LOAD

Description. This core load changes the run time for a grade upon operator request.

System Variables Referenced. IENDT, JOBN, DAY, SW3

Files Referenced.
File 3, Record 1

Programs Called. None

Calling Programs. None

System Subroutines Called. CONVR

Restart Core Load. N/A

41

Listing No.

Program. CPJSP

Type. INTERRUPT CORE LOAD

Description. This core load changes the sequence of grades upon operator request.

System Variables Referenced. JOBN, DAY, IENDT, SW3

Files Referenced.
File 3, Record 1

Programs Called. None

Calling Programs. None

System Subroutines Called. CONVR, PTIME

Restart Core Load. N/A

42

Listing No.

Program. STRND

Type. INTERRUPT CORE LOAD

Description. This core load initiates a trend log of the point specified by the operator.

System Variables Referenced. IPONT, IPERD, ITCNT

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. CONVR

Restart Core Load. N/A

43

Listing No.

Program. AIMON

Type. INTERRUPT CORE LOAD

Description. This core load logs all information about any point in the system.

System Variables Referenced. A, B, DAY, LOW, RANGE, SETPT

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. PTIME, CONVR

Restart Core Load. N/A

Listing No. 45

Program. SPECL

Type. NONPROCESS CORE LOAD

Description. This is a special one-time-only core load to set up the job files on disk for test purposes.

System Variables Referenced. AHL, ALL, A, B, RANGE, LOW, SETPT (1-8) JOBN

Files Referenced.
File 1, Record 1 - RANGE, LOW, A, B
File 2, Record 1 - J, ITIME, SETPT, AHL, ALL

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 46

Program. SCALB

Type. NONPROCESS CORE LOAD

Description. This nonprocess core load is for calibrating the set point stations.

System Variables Referenced. RANGE, LOW, A, B

Files Referenced.
File 1, Record 1

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 48

Program. RCALB

Type. NONPROCESS CORE LOAD

Description. This nonprocess core load is for calibrating the analog input points for Op-guide.

System Variables Referenced. A, B, RANGE, LOW

Files Referenced.
File 1, Record 1

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

Listing No. 50

Program. CMIPT

Type. NONPROCESS CORE LOAD

Description. This nonprocess core load is for calibrating the data entry dials.

System Variables Referenced. G(1-16), H(1-16)

Files Referenced.
File 1, Record 2

Programs Called. None

Calling Programs. None

System Subroutines Called. GETVL

Restart Core Load. N/A

Listing No. 52

Program. LOADJ

Type. NONPROCESS CORE LOAD

Description. This program loads the process job files on disk with data read from cards.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A

PROGRAM LISTING NO. 9: SYSTEM GENERATION

①

```
TASK 1800 TSX-II-1  SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER
 TASK DISK WRITE ADDRESSES PROGRAM
ENTER NO. TRIES ON DATA SW MAX001F
 DATA SWITCHES EQUAL LOGICAL DRIVE
 DRIVE CODES--HEX 0000 0001 0002
 THERE ARE NO DEFECTIVE CYLINDERS
 SEN SW 0 ON GO TO TASK OFF REDO
 TASK DISK WRITE ADDRESSES PROGRAM
ENTER NO. TRIES ON DATA SW MAX001F
 DATA SWITCHES EQUAL LOGICAL DRIVE
 DRIVE CODES--HEX 0000 0001 0002
 CYLINDERS  0013  0013  0013
    ARE DEFECTIVE
DO NOT USE SKEL.BLD WITH THIS PACK
 SEN SW 0 ON GO TO TASK OFF REDO
TASK 1800 TSX-II-1  SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER
```

②

```
//SYSTEMLOADER
//* IBM 1800 TSX-II SAMPLE SYSTEM
*ASSIGNMENT
00 01 33
01 02 04,33
02 02 00,08
03 02 02/02,33
04 03 01/01,36/04,37/05
05 04 33,06/03,11,12
06 03 10,34,33
07 04 03/07,32,20,16
08 01 33
09 01 33
10 01 33
11 01 33
12 01 33
13 01 33
14 01 33
15 01 33
99 01 42/06
DEVICE    LEV    BIT    IAC    LUN
PISW       00     00     33
DISK-1     01     00     04
PISW       01     01     33
TIMERS     02     00     00
DISK-2     02     01     08
CARD-1     03     00     02     02
PISW       03     01     33
TYP1G1     04     00     01     01
TYP2G1     04     01     36     04
TYP3G1     04     02     37     05
PISW       05     00     33
PRNT-1     05     01     06     03
DINP       05     02     11
DAOP       05     03     12
ADC-1      06     00     10
COMP-1     06     01     34
PISW       06     02     33
PAPTPE     07     00     03     07
CONSOL     07     01     32
RPQ-01     07     02     20
ADC-2      07     03     16
PISW       08     00     33
PISW       09     00     33
PISW       10     00     33
PISW       11     00     33
PISW       12     00     33
PISW       13     00     33
PISW       14     00     33
PISW       15     00     33
KEYB-1                   42     06
 YOU  DEFINED   000016 I/O DEVICES
 AND A TOTAL OF 000029 ILSW BITS
*LDDSK .LET
 SECTOR 0155
*LDDSK .DCOM
 SECTOR 0000
*LDDSK .MBT
 SECTOR 0002
```

② Continued

```
*LDDSK .SUP
 SECTOR 0005
*LDDSK .CLB
 SECTOR 0010
*LDDSK /CLST
 SECTOR 063A
*LDDSK .DUP
 SECTOR 001A
*LDDSK .ASM
 SECTOR 0066
*LDDSK .FOR
 SECTOR 0095
*LDDSK .SIM
 SECTOR 00F6
*LDDSK .EPRG
 SECTOR 0618
*LDDSK SBRT
IAND
CLEAR
CLOCK
COUNT
DMP   DMPHX DMPDC
DMPS  DMPST
DPART
ENDTS
IEOR
LD
LEVEL
MASK
OPMON
IOR
QIFON
QUEUE
RESMK
SAVMK
SETCL
TIMER
UNMK
UNQ
VIAQ
CONHX
TRPRT
FLIP
EADD  ESUB  EADDX ESUBX ESBR  ESBRX
EATN  EATAN
EAVL  EABS
EAXB  EAXBX
EAXI  EAXIX
EDVR  EDVRX EDIV  EDIVX
ELD   ELDX  ESTO  ESTOX
ELN   EALOG
EMPY  EMPYX
ESINE ESIN  ECOSN ECOS
ESQR  ESQRT
ETNH  ETANH
ETRTN ETNTR
EXPN  EEXP
FSBR  FSBRX FADD  FSUB  FADDX FSUBX
FARC
FATN  FATAN
FAVL  FABS
FAXB  FAXBX
FAXI  FAXIX
FBTD  FDTB
FDIV  FDIVX FDVR  FDVRX
FIXIX FIXI
FLD   FLDX  FSTO  FSTOX
FLN   FALOG
FLOAT
FMPY  FMPYX
FSINE FSIN  FCOSN FCOS
FSQR  FSQRT
FTNH  FTANH
FTRTN FTNTR
FXPN  FEXP
IABS
IFIX
NORM
SNR
XDD
XMD
XMDS
XSQR
BINDC
BINHX
```

(2) Continued

```
DCBIN
EBPA
EBPRT
HOLEB
HOLPR
HXBIN
PAPEB
PAPHL
PAPPR
PRT
ADRCK
COMGO COMG1
DATSW
DVCHK
ESIGN
FCTST
FSIGN
IOU
ISIGN
ISTOX
LDFAC STFAC SBFAC DVFAC
MDFIO MDAF  MDAI  MDCOM MDF   MDFX  MDI   MDIX  MDRED MDWRT
MDFND
MFIO  MRED  MWRT  MCOMP MIOAF MIOAI MIOFX MIOIX MIOF  MIOI
MGOTO MFIF  MIIF  MEIF
MIAR  MIARX MFAR  MFARX MEAR  MEARX
OVERF
PAUSE
REWND BCKSP EOF
SAVE  IOFIX
SLITE SLITT
SSWTC
STOP
SUBIN
SUBSC
TSTOP
TSTRT
TTEST TSET
UFIO  URED  UWRT  UIOI  UIOF  UIOAI UIOAF UIOFX UIOIX UCOMP
PLOTX
CARDN
PAPTN
MAGT
AIPTN AIPN
AISQN AISN
AIRN
ANINT
DINP
DIEXP
DICMP
DAOP
IOPE  OUSLY ETS
XSAVE XEXIT XLOAD
GAGED UNGAG
AIP
AIS
AIR
CS    VS    DI    PI
CSC   VSC   DIC   PIC
CSX   VSX   DIX   PIX
DAC   CO    DO    PO
QZERQ
QZ010
BT1BT
BT2BT
FCHAR
SCALF
FGRID
FPLOT
ECHAR
SCALE
EGRID
EPLOT
POINT
FCHRX FCHRI WCHRI
FRULE FMOVE FINC
ECHRX ECHRI VCHRI
ERULE EMOVE EINC
XYPLT
PLOTI PLOTS
//* SYDIR
*DEDIT 32K 020CYL
THE SOURCE CORE-SIZE IS   032768
THE OBJECT CORE-SIZE IS   032768
END SYSTEM LOAD
```

(2) Continued

```
TASK 1800 TSX-II-1  SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER
```

(3)

```
// JOB
// *  DEFINE THE SAMPLE SYSTEM TO BE A TWO DRIVE SYSTEM
// DUP
*DEFINE NDISK 2
DUP FUNCTION COMPLETED
```

(4)

```
// JOB
// *  LABEL DISK DRIVE ZERO WITH 00000
// DUP
*DLABL  0  00000
DUP FUNCTION COMPLETED
```

(5)

```
// JOB
// *  LABEL DISK DRIVE ONE WITH 11111
// DUP
*DLABL  1  11111
DUP FUNCTION COMPLETED
```

(6)

```
// JOB   X    X
// *  DEFINE DRIVE ZERO CONFIGURATION
// DUP
*DEFINE CONFG COPO                          020  000
DUP FUNCTION COMPLETED
```

(7)

```
// JOB   X    X
// *  DEFINE DRIVE ONE CONFIGURATION
// DUP
*DEFINE CONFG SX1N1I1DE1M1F1P1 16000 16767  080  010 05
DUP FUNCTION COMPLETED
```

(8)

```
// JOB          A
// *  SET UP FILES FILE1, FILE2, AND FILE3 ON DRIVE ONE
// DUP
*STOREDATAD   1    1 FILE1    002
DUP FUNCTION COMPLETED
*STOREDATAD   1    1 FILE2    100
DUP FUNCTION COMPLETED
*STOREDATAD   1    1 FILE3    003
DUP FUNCTION COMPLETED
```

(9)

```
// JOB          A
// *  STORE SYSTEM DIRECTOR FROM CARDS ON DISK
// DUP
*STORE      RD     1 SYDIR
SYDIR OUTTR CHAIN INTEX SHARE SPECL BACK  EACLK
DUP FUNCTION COMPLETED
```

(10)

```
// JOB         A
// *  INSKEL CALL COUNT SUBROUTINE
// FOR
*LIST ALL
**    PERIODIC PROGRAM SCHEDULER
      SUBROUTINE SCHED
C
C     THIS SUBROUTINE SCHEDULES THE PERIODIC EXECUTION OF THE PROGRAMS
C     LISTED IN THE EXTERNAL STATEMENT.  THIS SUBROUTINE IS ENTERED
C     EVERY 20 SECONDS.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      EXTERNAL SCAN2,LOG15,LOG60,SHIFT,WEEK
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
C
C     SET UP NEXT ENTRY INTO THE SCHEDULER.
C
      CALL COUNT(0,1,20)
C
C     INITIATE CLOSED LOOP CONTROL PROGRAM ON LEVEL 10.
C
      GO TO (97,1),SW3
   97 GO TO (99,98),SW0
   99 CALL LEVEL(10)
   98 SW4=1
C
C     TEST FOR TWO MINUTE LIMIT SCAN
C
      SW5=SW5+1
      GO TO (1,1,1,1,1,2),SW5
    2 CALL QUEUE(SCAN2,9,0)
      CALL ENDTS
      SW5=0
    1 CONTINUE
C
C     READ CLOCK
C
      CALL CLOCK(I)
C
C     TEST FOR FIFTEEN MINUTE LOG
C
      IF(IBASE)4,4,5
    4 IF(250-I)10,6,6
    5 IF(IBASE-I)6,6,10
    6 GO TO (66,67),SW3
   66 CALL QUEUE(LOG15,11,0)
      CALL ENDTS
   67 IBASE=IBASE+250
      IF(IBASE-23760)10,7,7
    7 IBASE=0
   10 CONTINUE
C
C     TEST FOR HOUR LOG
C
      IF(IBASZ)11,11,12
   11 IF(250-I)20,13,13
   12 IF(IBASZ-I)13,13,20
   13 GO TO (113,114),SW3
  113 CALL QUEUE(LOG60,12,0)
      CALL ENDTS
  114 IBASZ=IBASZ+1000
      IF(23100-IBASZ)14,14,20
   14 IBASZ=0
   20 CONTINUE
C
C     TEST FOR SHIFT END LOG AT 8.15,16.15,00.15
C
      IF(IBAZZ-250)21,21,22
   21 IF(I-1000)22,22,30
   22 IF(IBAZZ-I)23,23,30
   23 CALL QUEUE(SHIFT,13,0)
      CALL ENDTS
      IBAZZ=IBAZZ+8000
      IF(17000-IBAZZ)24,24,30
   24 IBAZZ=250
   30 CONTINUE
C
C     UPDATE DAY OF WEEK
C
      IF(I-100)31,31,35
   31 IF(SW1)32,32,36
   32 DAY=DAY+1
      IF(8-DAY)33,33,34
```

(10) Continued

```
   33 DAY=1
   34 SW1=1
      GO TO 36
   35 SW1=-1
   36 CONTINUE
C
C     TEST FOR 8.30 MONDAY MORNING LOG
C
      IF(DAY-2)100,40,100
   40 IF(I-8500)41,42,42
   41 SW2=-1
      GO TO 100
   42 IF(SW2)43,43,100
   43 CALL QUEUE(WEEK,14,0)
      SW2=1
      CALL ENDTS
  100 RETURN
      END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA

STATEMENT ALLOCATIONS
 97    =0021  99    =0027  98    =002A  2     =003E  1     =004A  4     =0051  5     =0059  6     =005F  66    =0065  67    =006D
 7     =0079  10    =007D  11    =0081  12    =0089  13    =008F  113   =0095  114   =009D  14    =00A9  20    =00AD  21    =00B3
 22    =00B9  23    =00BF  24    =00D3  30    =00D7  31    =00DD  32    =00E1  33    =00ED  34    =00F1  35    =00F7  36    =00FC
 40    =0102  41    =0108  42    =010F  43    =0113  100   =011F

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 SCAN2   LOG15   LOG60   SHIFT   WEEK    COUNT   LEVEL   QUEUE   ENDTS   CLOCK   COMGO

INTEGER CONSTANTS
     0=0002       1=0003      20=0004      10=0005       9=0006     250=0007      11=0008  23760=0009      12=000A    1000=000B
 23100=000C      13=000D    8000=000E   17000=000F     100=0010       8=0011       2=0012   8500=0013      14=0014

CORE REQUIREMENTS FOR SCHED
 COMMON       0  INSKEL COMMON     464  VARIABLES       2  PROGRAM     288


 END OF COMPILATION



SCHED
DUP FUNCTION COMPLETED
*DELET              SCHED
SCHED
D25 NAME NOT IN L/F
*STORE             1 SCHED
SCHED
DUP FUNCTION COMPLETED
```

(11)

```
// JOB        A
// *  INSKEL INTERRUPT SUBROUTINE
// FOR
*LIST ALL
**    LEVEL 10 SUBROUTINE FOR CLOSED LOOP CONTROL
      SUBROUTINE LEV10
C
C     THIS SUBROUTINE DOES CLOSED LOOP CONTROL OF EIGHT SET POINT
C     STATIONS.  EVERY TWENTY SECONDS THE SCHEDULER SUBROUTINE GIVES
C     A CALL LEVEL TO THIS ROUTINE AND SETS SW4 TO POINT TO THE SCAN
C     SECTION SUCH THAT ALL EIGHT POINTS ARE SCANNED.  WHEN THE SCAN IS
C     FINISHED THE OUTPUT TO EACH ON-LINE STATION IS COMPUTED AND THE
C     FIRST PULSE OUTPUT IS GIVEN.  TIMER B IS THEN USED TO SET UP
C     FIFTEEN MILLISECOND ENTRIES INTO THE SUBROUTINE SO THAT ALL
C     NECESSARY PULSES ARE GIVEN.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      EXTERNAL SOUT
      DIMENSION IBIT(8),IBIZ(8),IBIA(8),IOUT(3)
      DIMENSION INV(17)
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      DATA IBIT/Z4000,Z1000,Z0400,Z0100,Z0040,Z0010,Z0004,Z0001/
      DATA IBIZ/Z8000,Z2000,Z0800,Z0200,Z0080,Z0020,Z0008,Z0002/
      DATA IBIA/Z3FFF,ZCFFF,ZF3FF,ZFCFF,ZFF3F,ZFFCF,ZFFF3,ZFFFC/
```

(11) Continued

```
C
C     READ DIGITAL INPUT SWITCHES TO DETERMINE WHICH STATIONS ARE
C     ON-LINE.
C
      INV(17)=64
      CALL CSX(01011,INV(1),INV(17))
      INV(1)=0
      DO 678 J=1,8
      K=J+8
      OFFLN(J)=INV(K)
  678 INV(1)=INV(1)+OFFLN(J)
      IF(INV(1))699,699,679
  699 IOUT(1)=0
      GO TO 12
  679 CONTINUE
C
C     BRANCH ON SW4 TO EITHER SCAN SECTION OR OUTPUT SECTION.
C
      GO TO (1000,3000),SW4
 1000 CONTINUE
C
C     SCAN SECTION
C
C
C     SET PULSE OUTPUT WORD TO ZERO
C
      IOUT(1)=0
      CALL AIS(02001,VALUE(6),VALUE(15),4096)
   99 CALL AIS(0,IV)
      GO TO (99,88),IV
   88 CONTINUE
      DO 100 J=1,8
      IF(OFFLN(J))1,100,1
    1 IF(SETPT(J))111,100,111
  111 VAL=SETPT(J)-VALUE(J+5)
      VAL=VAL*2000./RANGE(J)
      COUMT(J)=VAL
      IF(COUMT(J))2,100,4
    2 COUMT(J)=-COUMT(J)
      IOUT(1)=IOR(IOUT(1),IBIT(J))
      GO TO 100
    4 IOUT(1)=IOR(IOUT(1),IBIZ(J))
  100 CONTINUE
      IF(IOUT(1))101,12,101
  101 SW4=2
 3000 CONTINUE
C
C     OUTPUT SECTION
C
      IOUT(2)=126
C
C     THE FOLLOWING BIT IS SET IN THE OUTPUT WORD FOR INCREMENTING A
C     PULSE COUNTER FOR DETERMINING THE NUMBER OF PULSES GIVEN AT
C     ANY TIME.
C
      IOUT(1)=IOR(IOUT(1),IBIZ(3))
      CALL PO(02001,IOUT(1),IOUT(3))
C
C     SEE WHICH POINTS ARE TO GO OUT NEXT TIME
C
      DO 10 J=1,8
      IF(OFFLN(J))3,9,3
    3 IF(COUMT(J))9,9,20
   20 COUMT(J)=COUMT(J)-1
      IF(COUMT(J))9,9,10
    9 IOUT(1)=IAND(IOUT(1),IBIA(J))
   10 CONTINUE
C
C     CALL TIMER B  FOR RE-ENTRY TO THIS SECTION IN 15 MILLISECONDS IF
C     THERE ARE ANY MORE PULSES TO GO OUT.
C
      IF(IOUT(1))11,12,11
   11 CALL TIMER(SOUT,2,15)
   12 CONTINUE
C
C     DISPLAY PULSE OUTPUT FOR VISUAL VERIFICATION OF DIRECTION OF
C     MOVEMENT.
C
      IOUT(2)=127
      CALL PO(01001,IOUT(1),IOUT(3))
      CALL INTEX
      END
```

(11) Continued

```
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18      A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54      H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30       CVAL(R )=0000        IBIT(I )=0009-0002  IBIZ(I )=0011-000A
  IBIA(I )=0019-0012  IOUT(I )=001C-001A    INV(I )=002D-001D       J(I )=002E           K(I )=002F          IV(I )=0030

STATEMENT ALLOCATIONS
 678  =0080  699  =009B  679  =00A3  1000 =00A9  99   =00C3  88   =00CD  1    =00DA  111  =00E3  2    =0114  4    =013B
 100  =0156  101  =0165  3000 =0169  3    =01A5  20   =01AE  9    =01BF  10   =01DA  11   =01E9  12   =01EF

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 SOUT    CSX     AIS      IOR      PO       IAND     TIMER    INTEX    FMPY     FDIVX    FLD      FSTO     IFIX     FLOAT    COMGO
 ISTOX   SUBSC

REAL CONSTANTS
  .200000E 04=0038

INTEGER CONSTANTS
    64=003A    1011=003B       0=003C       1=003D       8=003E    2001=003F    4096=0040       2=0041     126=0042      15=0043
   127=0044    1001=0045

CORE REQUIREMENTS FOR LEV10
 COMMON       0  INSKEL COMMON    464  VARIABLES     56  PROGRAM    466


 END OF COMPILATION


LEV10
DUP FUNCTION COMPLETED
// DUP
*DELET              LEV10
LEV10
D25 NAME NOT IN L/F
*STORE            1 LEV10
LEV10
DUP FUNCTION COMPLETED
```

(12)

```
// JOB         A
// *  INSKEL CALL TIMER SUBROUTINE
// FOR
*LIST ALL
**     TIMER B SUBROUTINE TO SETUP RE-ENTRY INTO LEVEL 10 PROGRAM
       SUBROUTINE SOUT
C
C      THIS SUBROUTINE SERVICES TIMER B AND IS USED FOR INITIATING
C      ENTRIES INTO THE LEVEL 10 SUBROUTINE FOR THE OUTPUTING OF PULSES
C      TO THE SET POINT STATIONS.
C
       INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       GO TO (3,2),SWO
     3 GO TO (2,1),SW4
     1 CALL LEVEL(10)
     2 RETURN
       END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18      A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54      H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30

STATEMENT ALLOCATIONS
 3     =0008  1     =000E  2     =0011

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 LEVEL   COMGO
```

(12) **Continued**

```
INTEGER CONSTANTS
     10=0000

CORE REQUIREMENTS FOR SOUT
 COMMON       0  INSKEL COMMON    464  VARIABLES       0  PROGRAM      20


 END OF COMPILATION


SOUT
DUP FUNCTION COMPLETED
*STORE            1 SOUT
SOUT
DUP FUNCTION COMPLETED
```

(13)

```
// JOB        A
// *  INSKEL INTERRUPT SUBROUTINE
// FOR
*LIST ALL
**     QUEUE 15 MINUTE LOG ON DEMAND
       SUBROUTINE QUE15
C
C      THIS SUBROUTINE QUEUES THE FIFTEEN MINUTE LOG ROUTINE ON DEMAND.
C
       EXTERNAL LOG15
       CALL ENDTS
       CALL QUEUE(LOG15,7,0)
       CALL INTEX
       END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 LOG15   ENDTS   QUEUE   INTEX

INTEGER CONSTANTS
      7=0000       0=0001

CORE REQUIREMENTS FOR QUE15
 COMMON       0  INSKEL COMMON      0  VARIABLES       0  PROGRAM      14


 END OF COMPILATION


QUE15
DUP FUNCTION COMPLETED
*STORE            1 QUE15
QUE15
DUP FUNCTION COMPLETED
```

(14)

```
// JOB        A
// *  INSKEL CALL COUNT SUBROUTINE
// FOR
*LIST ALL
**     PERIODIC QUEUE OF TREND LOG SUBROUTINE
       SUBROUTINE TCONT
C
C      THIS SUBROUTINE PERIODICALLY QUEUES THE TREND LOG PROGRAM THE
C      NUMBER OF TIMES SPECIFIED.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       EXTERNAL TREND
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       CALL ENDTS
       CALL QUEUE(TREND,2,0)
       ITCNT=ITCNT-1
       IF(ITCNT)2,2,1
```

(14) Continued

```
    1 CALL COUNT(2,3,IPERD)
    2 RETURN
      END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30

STATEMENT ALLOCATIONS
 1     =0017  2     =001C

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 TREND   ENDTS   QUEUE   COUNT

INTEGER CONSTANTS
     2=0000       0=0001       1=0002       3=0003

CORE REQUIREMENTS FOR TCONT
 COMMON       0  INSKEL COMMON    464  VARIABLES       0  PROGRAM     30


 END OF COMPILATION



TCONT
DUP FUNCTION COMPLETED
*DELET              TCONT
TCONT
D25 NAME NOT IN L/F
*STORE            1 TCONT
TCONT
DUP FUNCTION COMPLETED
```

(15)

```
// JOB        A
// *  INSKEL INTERRUPT SUBROUTINE
// FOR
*LIST ALL
**     ABORT TREND LOG SUBROUTINE
       SUBROUTINE TABRT
C
C      THIS SUBROUTINE ABORTS THE TREND LOG ON DEMAND.
C
       INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       ITCNT=0
       CALL INTEX
       END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 INTEX

INTEGER CONSTANTS
     0=0000

CORE REQUIREMENTS FOR TABRT
 COMMON       0  INSKEL COMMON    464  VARIABLES       0  PROGRAM      8


 END OF COMPILATION
```

(15) Continued

```
TABRT
DUP FUNCTION COMPLETED
*STORE            1 TABRT
TABRT
DUP FUNCTION COMPLETED
```

(16)

```
// JOB         A
// *  USER SUBROUTINE
// FOR
*LIST ALL
**     SUBROUTINE FOR READING DATA ENTRY DIALS
       SUBROUTINE GETVL(INVAL)
C
C      THIS SUBROUTINE READS THE ANALOG INPUT VALUE FOR EACH OF THE
C      SIXTEEN DATA ENTRY DIALS.
C
       DIMENSION INS(12),INR(12),INVAL(16)
       CALL AIS(02011,INR(1),INR(10),0)
       CALL AIS(02001,INS(1),INS(12),4098)
     1 CALL AIS(00010,IV)
       GO TO (1,2),IV
     2 CALL AIS(0,IV)
       GO TO(2,3),IV
     3 DO 4 J=1,8
       K=9-J
     4 INVAL(J+8)=INR(K)
       DO 5 J=1,4
       K=11-J
       INVAL(J)=INS(K)
     5 INVAL(J+4)=INS(K-6)
       RETURN
       END
VARIABLE ALLOCATIONS
   INS(I )=000B-0000   INR(I )=0017-000C    IV(I )=0018          J(I )=0019          K(I )=001A

STATEMENT ALLOCATIONS
 1     =0059  2     =0063  3     =006D  4     =0077  5     =00AA

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 AIS      COMGO    ISTOX    SUBSC    SUBIN

INTEGER CONSTANTS
  2011=0020      0=0021    2001=0022    4098=0023     10=0024      1=0025      8=0026      9=0027      4=0028      11=0029

CORE REQUIREMENTS FOR GETVL
 COMMON      0  INSKEL COMMON      0  VARIABLES     32  PROGRAM    166


 END OF COMPILATION


GETVL
DUP FUNCTION COMPLETED
*STORE            1 GETVL
GETVL
DUP FUNCTION COMPLETED
```

(17)

```
// JOB         A
// *  USER SUBROUTINE
// FOR
*LIST ALL
**     SUBROUTINE FOR READING AND CONVERTING DATA ENTRY DIALS
       SUBROUTINE CONVR(INVAL)
C
C      THIS SUBROUTINE SCANS THE 16 DATA ENTRY DIALS AT THE PROCESS
C      OPERATORS CONSOLE AND CONVERTS THEM TO AN INTEGER VALUE WITH
C      RANGE 0 TO 9.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INVAL(16)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
```

(17) Continued

```
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H
      CALL GETVL (INVAL)
      DO 10 J=1,16
   10 INVAL(J)=INVAL(J)*G(J)+H(J)
      RETURN
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34     J(I )=0000

STATEMENT ALLOCATIONS
 10    =0014

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 GETVL    FADDX    FMPYX    IFIX     FLOAT    ISTOX    SUBSC    SUBIN

INTEGER CONSTANTS
     1=0004      16=0005

CORE REQUIREMENTS FOR CONVR
 COMMON       0  INSKEL COMMON    460   VARIABLES       4   PROGRAM      54


 END OF COMPILATION


CONVR
DUP FUNCTION COMPLETED
*STORE            1 CONVR
CONVR
DUP FUNCTION COMPLETED
```

(18)

```
// JOB         A
// *  USER SUBROUTINE
// FOR
*LIST ALL
**     TIME CONVERSION SUBROUTINE
       SUBROUTINE PTIME(X)
C
C      THIS SUBROUTINE READS THE CLOCK AND CONVERTS THE TIME TO A
C      FLOATING POINT NUMBER WITH THE DECIMAL POINT SEPERATING HOURS
C      AND MINUTES.
C
       CALL CLOCK(I)
       J=I/1000*1000
       I=I-J
       X=((I*60./1000.)+J/10)/100.
       RETURN
       END
VARIABLE ALLOCATIONS
     I(I )=0002           J(I )=0003

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 CLOCK    FADD     FMPY     FDIV     FSTO     FLOAT    SUBIN

REAL CONSTANTS
  .600000E 02=0004    .100000E 04=0006    .100000E 03=0008

INTEGER CONSTANTS
  1000=000A     10=000B

CORE REQUIREMENTS FOR PTIME
 COMMON       0  INSKEL COMMON       0  VARIABLES       4   PROGRAM      56


 END OF COMPILATION


PTIME
DUP FUNCTION COMPLETED
```

(18) **Continued**

```
// DUP
*STORE            1 PTIME
PTIME
DUP FUNCTION COMPLETED
```

(19)

```
// JOB        A
// *  USER SUBROUTINE
// *  THE FOLLOWING TWO SUBROUTINES ARE USED IN FORTRAN TO OBTAIN THE
// *  ADDRESS OF EITHER A VARIABLE OR A SUBROUTINE
// ASM
  *LIST
  *PRINT SYMBOL TABLE
```

```
                     **************************************************
                     *             IADDR SUBROUTINE                   *
                     **************************************************
                     *                                                *
                     *     THIS SUBROUTINE GETS THE ADDRESS OF A      *
                     *     FORTRAN VARIABLE.                          *
                     *                                                *
                     *     FORTRAN CALL                               *
                     *             I=IADDR(ABC)                       *
                     *                                                *
                     *     AFTER EXECUTION OF THE ABOVE STATEMENT     *
                     *     I EQUALS THE ADDRESS OF THE VARIABLE ABC.  *
                     *                                                *
                     *     ASM GENERATED CODE                         *
                     *             CALL        IADDR                  *
                     *             DC          ADDR(ABC)              *
                     *                                                *
                     *     RESULT IS IN THE ACCUM UPON RETURN         *
                     *                                                *
                     *     THE SUBROUTINE IS RE-ENTRANT               *
                     *                                                *
                     **************************************************
0000     09044119          ENT      IADDR      DEFINE ENTRY POINT
0000 0   0000        IADDR DC       0          CALL ENTRY
0001 01  66800000          LDX   I2 IADDR      SAVE RET ADDR AND SET XR2
0003 0   C200              LD     2 0          TO PARAMETER--GET PARAMETR
0004 00  4E000001          BSC   L2 1          RETURN
0006                       END
```

```
                                   SYMBOL TABLE


      IADDR 0000

      NO ERRORS IN ABOVE ASSEMBLY.
IADDR
DUP FUNCTION COMPLETED
// DUP
*STORE            1 IADDR
IADDR
DUP FUNCTION COMPLETED
// ASM
  *LIST
  *PRINT SYMBOL TABLE
```

(19) Continued

```
                              **************************************************
                              *               ISBAD SUBROUTINE                 *
                              **************************************************
                              *                                                *
                              *     THIS SUBROUTINE GETS THE ADDRESS OF THE    *
                              *     ENTRY POINT TO A SUBROUTINE.               *
                              *                                                *
                              *     FORTRAN CALL                               *
                              *             EXTERNAL SUBR                      *
                              *              .                                 *
                              *              .                                 *
                              *              .                                 *
                              *              .                                 *
                              *             I=ISBAD(SUBR)                      *
                              *                                                *
                              *     AFTER EXECUTION OF THE ABOVE STATEMENT     *
                              *     I EQUALS THE ADDRESS OF THE ENTRY POINT    *
                              *     OF THE SUBROUTINE SUBR.                    *
                              *                                                *
                              *     ASM GENERATED CODE                         *
                              *             CALL       ISBAD                   *
                              *             CALL       SUBR                    *
                              *                                                *
                              *     ENTRY ADDR TO SUB IS IN ACCUM UPON RETURN  *
                              *                                                *
                              *     THE SUBROUTINE IS RE-ENTRANT               *
                              *                                                *
                              **************************************************
0000     09882044                   ENT      ISBAD      DEFINE ENTRY POINT
0000 0   0000               ISBAD DC         0          CALL ENTRY POINT
0001 01  66800000                   LDX  I2  ISBAD      XR2#RET ADDR AND CALL PARA
0003 0   C200                       LD     2 0          IS  IT BSI L OR BSI I
0004 0   1008                       SLA      8          PUT INDIRECT BIT IN 0 POS
0005 01  4C10000A                   BSC  L   *+3,-      BRANCH FOR DIRECT BSI
0007 00  C6800001                   LD   I2  1          GET SUB ENTRY POINT
0009 0   7001                       MDX      *+1
000A 0   C201                       LD     2 1          GET SUB ENTRY POINT
000B 00  4E000002                   BSC  L2  2          EXIT
000E                                END


                                         SYMBOL TABLE


      ISBAD 0000

      NO ERRORS IN ABOVE ASSEMBLY.
ISBAD
DUP FUNCTION COMPLETED
// DUP
*STORE            1 ISBAD
ISBAD
DUP FUNCTION COMPLETED
```

(20)

```
// JOB         A
// *  INSKEL INTERRUPT SUBROUTINE
// FOR
*LIST ALL
**    QUEUE OF CE UNMASK ROUTINE
      SUBROUTINE    CESET
C
C     THIS SUBROUTINE QUEUES THE CE UMASK PROGRAM SO THAT DEVICES MAY
C     BE TAKEN OFF-LINE OR PUT ON-LINE.
C
      EXTERNAL CEINT
      CALL ENDTS
      CALL QUEUE(CEINT,20,0)
      CALL INTEX
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 CEINT    ENDTS    QUEUE    INTEX

INTEGER CONSTANTS
    20=0000       0=0001
```

(20) Continued

```
CORE REQUIREMENTS FOR CESET
 COMMON       0  INSKEL COMMON       0  VARIABLES       0  PROGRAM      14


 END OF COMPILATION


CESET
DUP FUNCTION COMPLETED
// DUP
*STORE              1 CESET
CESET
DUP FUNCTION COMPLETED
```

(21)

```
// JOB         A
// *  INSKEL INTERRUPT SUBROUTINE
// FOR
*LIST ALL
**     ABORT GRADE PROCESS INTERRUPT SUBROUTINE
       SUBROUTINE ABORT
C
C      THIS SUBROUTINE QUEUES THE GRADE CHANGE PROGRAM CAUSING THE
C      PRESENT GRADE TO BE ABORTED.
C
       EXTERNAL      GRADE
       CALL ENDTS
       CALL QUEUE(GRADE,5,0)
       CALL INTEX
       END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 GRADE   ENDTS   QUEUE   INTEX

INTEGER CONSTANTS
     5=0000       0=0001

CORE REQUIREMENTS FOR ABORT
 COMMON       0  INSKEL COMMON       0  VARIABLES       0  PROGRAM      14


 END OF COMPILATION


ABORT
DUP FUNCTION COMPLETED
// DUP
*STORE              1 ABORT
ABORT
DUP FUNCTION COMPLETED
```

(22)

```
// JOB         A
// *  INSKEL CALL COUNT SUBROUTINE
// FOR
*LIST ALL
**     END OF GRADE COUNT SUBROUTINE
       SUBROUTINE ENDGD
C
C      THIS SUBROUTINE ABORTS THE GRADE IN PROGRESS WHEN THE RUN TIME FOR
C      THAT GRADE HAS ELAPSED.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       EXTERNAL      GRADE
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       GO TO (1,2),SW3
     1 CONTINUE
       CALL ENDTS
       CALL QUEUE(GRADE,5,0)
     2 RETURN
       END
```

(22) Continued

```
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18      A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75           G(R*)=FE72-FE54      H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30

STATEMENT ALLOCATIONS
 1     =0009  2     =0011

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 GRADE    ENDTS    QUEUE    COMGO

INTEGER CONSTANTS
     5=0000        0=0001

CORE REQUIREMENTS FOR ENDGD
 COMMON       0  INSKEL COMMON     464  VARIABLES       0  PROGRAM      20       .


 END OF COMPILATION


ENDGD
DUP FUNCTION COMPLETED
// DUP
*STORE              1 ENDGD
ENDGD
DUP FUNCTION COMPLETED
```

(23)

```
// JOB          A
// DUP
*DUMPLET



      LET


PACK LABEL
 00000

.DCOM  0010   0000    .MBT   0020   0010    .SKSB  0020   0030    .SUP   00B0   0050    .CLB   00A0   0100    .DUP   0440   01A0
.ASM   0300   05E0    .FOR   0680   08E0    .SIM   05F0   0F60    .LET   0080   1550    IAND   0002   15D0    CLEAR  0009   15D2
CLOCK  0002   15DB    COUNT  0004   15DD    DMP    0017   15E1    DMPHX                 DMPDC                 DMPS   0010   15F8
DMPST                 DPART  0002   1608    ENDTS  0002   160A    IEOR   0002   160C    LD     0002   160E    LEVEL  0004   1610
MASK   0003   1614    OPMON  0002   1617    IOR    0002   1619    QIFON  000A   161B    QUEUE  000C   1625    RESMK  0004   1631
SAVMK  0003   1635    SETCL  0003   1638    TIMER  0006   163B    UNMK   0005   1641    UNQ    0005   1646    VIAQ   0007   164B
CONHX  0006   1652    TRPRT  0007   1658    FLIP   0007   165F    EADD   000B   1666    ESUB                  EADDX
ESUBX                 ESBR                  ESBRX                 EATN   000D   1671    EATAN                 EAVL   0003   167E
EABS                  EAXB   0006   1681    EAXBX                 EAXI   0006   1687    EAXIX                 EDVR   0007   168D
EDVRX                 EDIV                  EDIVX                 ELD    0009   1694    ELDX                  ESTO
ESTOX                 ELN    000B   169D    EALOG                 EMPY   0004   16A8    EMPYX                 ESINE  000D   16AC
ESIN                  ECOSN                 ECOS                  ESQR   0007   16B9    ESQRT                 ETNH   0006   16C0
ETANH                 ETRTN  0004   16C6    ETNTR                 EXPN   000B   16CA    EEXP                  FSBR   000B   16D5
FSBRX                 FADD                  FSUB                  FADDX                 FSUBX                 FARC   0004   16E0
FATN   000C   16E4    FATAN                 FAVL   0003   16F0    FABS                  FAXB   0006   16F3    FAXBX
FAXI   0006   16F9    FAXIX                 FBTD   001A   16FF    FDTB                  FDIV   0008   1719    FDIVX
FDVR                  FDVRX                 FIXIX  0005   1721    FIXI                  FLD    0009   1726    FLDX
FSTO                  FSTOX                 FLN    000B   172F    FALOG                 FLOAT  0003   173A    FMPY   0005   173D
FMPYX                 FSINE  000B   1742    FSIN                  FCOSN                 FCOS                  FSQR   0007   174D
FSQRT                 FTNH   0006   1754    FTANH                 FTRTN  0004   175A    FTNTR                 FXPN   0009   175E
FEXP                  IABS   0003   1767    IFIX   0004   176A    NORM   0004   176E    SNR    0003   1772    XDD    0006   1775
XMD    0005   177B    XMDS   0004   1780    XSQR   0004   1784    BINDC  0006   1788    BINHX  0004   178E    DCBIN  0006   1792
EBPA   0006   1798    EBPRT  000A   179E    HOLEB  0012   17A8    HOLPR  000D   17BA    HXBIN  0005   17C7    PAPEB  0010   17CC
PAPHL  0014   17DC    PAPPR  0011   17F0    PRT    0005   1801    ADRCK  0007   1806    COMGO  0006   180D    COMG1
DATSW  0004   1813    DVCHK  0002   1817    ESIGN  0005   1819    FCTST  0003   181E    FSIGN  0005   1821    IOU    0007   1826
ISIGN  0003   182D    ISTOX  0003   1830    LDFAC  0004   1833    STFAC                 SBFAC                 DVFAC
MDFIO  0023   1837    MDAF                  MDAI                  MDCOM                 MDF                   MDFX
MDI                   MDIX                  MDRED                 MDWRT                 MDFND  0008   185A    MFIO   0059   1862
MRED                  MWRT                  MCOMP                 MIOAF                 MIOAI                 MIOFX
MIOIX                 MIOF                  MIOI                  MGOTO  000E   18BB    MFIF                  MIIF
MEIF                  MIAR   000E   18C9    MIARX                 MFAR                  MFARX                 MEAR
MEARX                 OVERF  0002   18D7    PAUSE  0002   18D9    REWND  0009   18DB    BCKSP                 EOF
SAVE   000A   18E4    IOFIX                 SLITE  0006   18EE    SLITT                 SSWTC  0004   18F4    STOP   0003   18F8
SUBIN  0005   18FB    SUBSC  0004   1900    TSTOP  0002   1904    TSTRT  0002   1906    TTEST  0003   1908    TSET
```

(23) Continued

```
UFIO   001C   1908    URED                 UWRT                 UIOI                 UIOF                 UIOAI
UIOAF                 UIOFX                UIOIX                UCOMP                PLOTX  000D   1927   CARDN  0016   1934
PAPTN  0010   194A    MAGT   0020   195A   AIPTN  0009   197A   AIPN                 AISQN  000F   1983   AISN
AIRN   000D   1992    ANINT  0014   199F   DINP   0013   19B3   DIEXP  0006   19C6   DICMP  0007   19CC   DAOP   0013   19D3
IOPE   0009   19E6    OUSLY                ETS                  XSAVE  0009   19EF   XEXIT                XLOAD
GAGED  0003   19F8    UNGAG                AIP    0004   19FB   AIS    000D   19FF   AIR    0011   1A0C   CS     0008   1A1D
VS                    DI                   PI                   CSC    000A   1A25   VSC                  DIC
PIC                   CSX    0004   1A2F   VSX                  DIX                  PIX                  DAC    0007   1A33
CO                    DO                   PO                   QZERQ  0002   1A3A   QZ010  0006   1A3C   BT1BT  0007   1A42
BT2BT  0003   1A49    FCHAR  0005   1A4C   SCALF  0002   1A51   FGRID  0007   1A53   FPLOT  0004   1A5A   ECHAR  0005   1A5E
SCALE  0002   1A63    EGRID  0008   1A65   EPLOT  0005   1A6D   POINT  0007   1A72   FCHRX  0024   1A79   FCHRI
WCHRI                 FRULE  0009   1A9D   FMOVE                FINC                 ECHRX  0025   1AA6   ECHRI
VCHRI                 ERULE  000B   1ACB   EMOVE                EINC                 XYPLT  0007   1AD6   PLOTI  0003   1ADD

PLOTS                 .TEMP  1AE0   1B00   .E     5A00   1B00


             FLET

PACK LABEL
 00000

9DUMY  00A0   05A0    .E     00A0   05A0


          LET

PACK LABEL
 11111

.LET   0080   0000    SYDIR  009E   0080   OUTTR                CHAIN                INTEX                SHARE
SPECL                 BACK                 EACLK                SCHED  0014   011E   LEV10  0024   0132   SOUT   0003   0156
QUE15  0002   0159    TCONT  0003   015B   TABRT  0002   015E   GETVL  000B   0160   CONVR  0005   016B   PTIME  0005   0170
IADDR  0002   0175    ISBAD  0002   0177   CESET  0002   0179   ABORT  0002   017B   ENDGD  0002   017D   .TEMP  017F   0180
.E     1180   0180



             FLET

PACK LABEL
 11111

.PRWS  0051   1118    .FIOS  000F   1169   .MESS  00A3   1178   /EPDM  7FFF   121B   /EPSV  0780   1282   /INSV  48FF   1288
/NPSV  4180   12C3    FILE1  0002   12F8   FILE2  0064   12FA   FILE3  0003   135E   9DUMY  0217   1361   /SPSV  4180   1578
/PRSV  4180   15AD    .SKEL  0036   15E2   .EPRG  0022   1618   /CLST  0780   163A   .E     0280   12F8

DUP FUNCTION COMPLETED


// JOB         A
// END OF ALL JOBS
```

(24)

```
TASK 1800 TSX-II-1  SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER
PLACE TASK DECK IN CARD HOPPER
PUT SKL BUILD PROG IN CARD HOPPER
// JOB         A
// XEQ SKBLD
*INCLDSCHED/2600,ENDGD/2601,QUE15/0001,CESET/2500,ABORT/0000,LEV10/2410
*INCLDTCONT/2602,TABRT/0002
*CCEND

KOC   ANINT 0023   LEV.1

SKEL CORE   MAP
TYPE NAME   ARG1   ARG2

LIBF DISKN  02A9   3EBB
LIBF TYPEN  0674   3EBE
LIBF WRTYN  0674   3EBE
LIBF PRNTN  0B6F   3EC1
```

24 Continued

```
LIBF CARDN 0E0D  3EC4
CALL EXIT  206E  00B6
CALL LINK  2070  008E
INSK       0F62  1349
PNT  SYDIR 134A  3DB4
ICI  SCHED 2391  1200
ICI  ENDGD 24A0  1201
ICI  QUE15 24B4  0001
ICI  CESET 24C2  1100
ICI  ABORT 24D0  0000
ICI  LEV10 2522  100A
ICI  TCONT 26EA  1202
ICI  TABRT 2705  0002
CALL OUTTR 1D8C  3E46
CALL CHAIN 1F63  3E45
CALL INTEX 1E90  3E44
CALL SHARE 2009  3E43
CALL SPECL 1FA9  3E42
CALL BACK  1FF1  3E41
CALL EACLK 2305  3E40
CALL COUNT 270C  3E3F
LIBF COMGO 2740  3EC7
CALL LEVEL 2792  3E3E
CALL QUEUE 27CA  3E3D
PNT  SCAN2       3DB8
CALL ENDTS 288C  3E3C
CALL CLOCK 2896  3E3B
PNT  LOG15       3DBC
PNT  LOG60       3DC0
PNT  SHIFT       3DC4
PNT  WEEK        3DC8
PNT  GRADE       3DCC
PNT  CEINT       3DD0
LIBF ISTOX 28A8  3ECA
CALL CSX   28C8  3E3A
LIBF SUBSC 28EE  3ECD
CALL AIS   291C  3E39
LIBF FLOAT 29E6  3ED0
LIBF FSTO  2A4A  3ED3
LIBF FLD   2A64  3ED6
LIBF FMPY  2A7D  3ED9
LIBF FDIVX 2ABA  3EDC
LIBF IFIX  2B26  3EDF
CALL IOR   2B52  3E38
CALL PO    2B60  3E37
CALL IAND  2BBC  3E36
CALL TIMER 2BCA  3E35
CALL SOUT  2C21  3E34
PNT  TREND       3DD4
LIBF COMG1 2781  3EE2
LIBF ADRCK 2C34  3EE5
CALL VSX   28C8  3E33
CALL DIX   28C8  3E32
CALL PIX   28C8  3E31
CALL QZ010 2C98  3E30
CALL QZERQ 2CEC  3E2F
LIBF DIEXP 2CFA  3EE8
LIBF AISQN 2D46  3EEB
LIBF NORM  2E3C  3EEE
LIBF FLDX  2A5F  3EF1
LIBF FSTOX 2A00  3EF4
LIBF FMPYX 2A78  3EF7
LIBF FARC  2E68  3EFA
LIBF FDIV  2ABF  3EFD
LIBF FDVR  2B05  3F00
LIBF FDVRX 2B00  3F03
CALL FTNTR 2E9C  3E2E
CALL FTRTN 2EB6  3E2D
CALL DAC   2B60  3E2C
CALL CO    2B60  3E2B
CALL DO    2B60  3E2A
LIBF DAOP  2EC4  3F06
CALL GAGED 2FF0  3E29
CALL UNGAG 3001  3E28
LIBF AISN  2D46  3F09
CALL ANINT 3010  3E27
PTCH       314C  3DB1
```

(24) Continued

```
  ICL TABLE MAP
LLBB  WC/EP  SA    ICLT

0000  24D0         1454
0001  24B4         1456
0002  2705         1458
100A  2522         15B8
1100  24C2         15C4
1200  2391         15CC
1201  24A0         15CE
1202  26EA         15D0

 K13  SCAN2 LEV.1


 K13  LOG15 LEV.1


 K13  LOG60 LEV.1


 K13  SHIFT LEV.1


 K13  WEEK  LEV.1


 K13  GRADE LEV.1


 K13  CEINT LEV.1


 K13  TREND LEV.1

 DATA SW 0  ON TO ABORT SKEL

 SKB, SYDIR LD NX

TASK 1800 TSX-II-1  SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
```

(25)

```
// JOB          A
// *  MAINLINE CORE LOAD
// FOR COLON
*LIST ALL
**    OFF-LINE COLD START FOR PERPETUAL TIME SHARING
*IOCS(DISK)
C
C      THIS COLD START CORE LOAD IS USED TO GIVE PERPETUAL TIME SHARING.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 1(2,320,U,II)
C
C      UNMASK ALL LEVELS AND CALL VIAQ WITH QUEUE EMPTY
C
       CALL UNMK(-1,-1)
       READ(1'1)RANGE,LOW,A,B
       READ(1'2)G,H
       CALL VIAQ
       END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30         II(I )=000A

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 UNMK     VIAQ     MDRED    MDCOM    MDAI     MDAF
```

25 Continued

```
INTEGER CONSTANTS
     1=000C       2=000D

CORE REQUIREMENTS FOR COLDN
 COMMON       0  INSKEL COMMON    464  VARIABLES      12  PROGRAM      44


 END OF COMPILATION



COLDN
DUP FUNCTION COMPLETED
// DUP
*DELET    M          COLDN DUM
DUM
D25 NAME NOT IN L/F
*STORECIL M        1 COLDN COLDN COLDN
*FILES(1,FILE1,1)
*CCEND


CLB, BUILD COLDN

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  000C
*IST TABLE 3F18  0036
*PNT TABLE 3F4E  0008
*DFT TABLE 3F56  0006
MAIN COLDN 3F64
PNT  COLDN 3F50
PNT  COLDN 3F54
CALL UNMK  3F8E
LIBF MDRED 4040  3F0C
LIBF MDAF  3FDF  3F0F
LIBF MDAI  3FEC  3F12
LIBF MDCOM 40A2  3F15
CALL VIAQ  434C
CALL BT2BT 43AC
CALL SAVE  43C8
CALL IOFIX 442C
CORE       445E  3BA2

CLB, COLDN LD XQ

DUP FUNCTION COMPLETED
```

26

```
// JOB        A
// *  MAINLINE CORE LOAD
// ASM COLDS
  *LIST
  *PRINT SYMBOL TABLE


                     ***************************************************
                     *                                                 *
                     *          THIS IS THE NORMAL COLD                *
                     *          START CORE LOAD.                       *
                     *                                                 *
                     *          IT SETS TIME SHARING TIME TO ZERO      *
                     *          SO THAT CONSOLE INTERRUPT              *
                     *          MUST HAVE BEEN PUSHED BEFORE           *
                     *          LOGICAL DRIVE ZERO IS EVER             *
                     *          REFERENCED.                            *
                     *                                                 *
                     *          THIS CORE LOAD CHAINS TO COLDP         *
                     *          TO ACTUALLY COLD START THE PROCESS     *
                     *                                                 *
                     ***************************************************
0000 0  1010         START SLA      16       SET TIME SHARING TIME TO 0
0001 00 D4000029           STO  L   41
0003 30 03201255           CALL     CHAIN
0005 30 03593117           CALL     COLDP       CHAIN TO COLDP
0008    0000               END      START
```

(26) Continued

```
                                        SYMBOL TABLE


     START 0000

     NO ERRORS IN ABOVE ASSEMBLY.
COLDS
DUP FUNCTION COMPLETED
// DUP
*STORECIL M        1 COLDS COLDS COLDS
*CCEND


  CLB, BUILD COLDS

  CORE LOAD  MAP
  TYPE NAME  ARG1  ARG2

  *CDW TABLE 3E82  000C
  *IBT TABLE 3E8E  001D
  *FIO TABLE 3EAB  0010
  *ETV TABLE 3EBB  0051
  *IST TABLE 3F0C  0036
  *PNT TABLE 3F42  000C
  MAIN COLDS 3F4E
  PNT  COLDS 3F44
  PNT  COLDS 3F48
  PNT  COLDP 3F4C
  CORE       3F58  40A8

  CLB, COLDS LD XQ

D 45 CORELOADS NOT FOUND
COLDP
DUP FUNCTION COMPLETED
```

(27)

```
// JOB          A
// *  MAINLINE CORE LOAD
// FOR RSTAR
*LIST ALL
**    RESTART CORE LOAD
*IOCS(DISK,TYPEWRITER)
C
C     THIS IS THE SYSTEM RESTART CORE LOAD.  WHEN EVER A RESTART
C     CONDITION OCCURS THIS ROUTINE IS LOADED TO VARIABLE CORE TO MAKE
C     SURE SYSTEM CONSTANTS IN INSKEL COMMON ARE VALID.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      EXTERNAL GRADE
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      DEFINE FILE 1(2,320,U,II)
      DEFINE FILE 3(3,320,U,II)
C     RESTORE INSKEL COMMON
      SW0=2
      CALL UNMK(-1,-1)
      CALL PTIME(TIME)
      WRITE(1,1)TIME
    1 FORMAT(' PROCESS RESTART       TIME'F7.2)
      READ(1'1)RANGE,LOW,A,B
      READ(3'1)JOBN,DAY,IENDT,SW3
      READ(3'2)SETPT
      READ(3'3)AHL,ALL
      GO TO (95,96),SW3
   96 CALL VIAQ
   95 CONTINUE
      SW0=1
      CALL COUNT(0,1,5)
      CALL CLOCK(I)
      IF (I-IENDT)101,102,103
  101 IPER=IENDT-I
      GO TO 104
  102 CALL CHAIN(GRADE)
  103 IPER=(24000-IENDT)+I
  104 AA=IPER*3.6
      IF(32000.-AA)102,105,105
```

27 Continued

```
  105 IPER=AA
      CALL COUNT(1,2,IPER)
      CALL VIAQ
      END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFCO OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30      CTIME(R )=000C         AA(R )=000E          II(I )=0014


STATEMENT ALLOCATIONS
 1    =0022  96   =0080  95   =0082  101  =0096  102  =009E  103  =00A2  104  =00AA  105  =00B8

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 GRADE   UNMK    PTIME   VIAQ    COUNT   CLOCK   CHAIN   FSUB    FMPY    FLD     FSTO    IFIX    FLOAT   COMGO   LDFAC
 MWRT    MCOMP   MIOF    MDRED   MDCOM   MDAI    MDAF    MDI     TYPEN   EBPRT

REAL CONSTANTS
  .360000E 01=0018    .320000E 05=001A

INTEGER CONSTANTS
     2=001C      1=001D      3=001E      0=001F       5=0020  24000=0021

CORE REQUIREMENTS FOR RSTAR
 COMMON      0  INSKEL COMMON    464  VARIABLES     24  PROGRAM    172


 END OF COMPILATION


RSTAR
DUP FUNCTION COMPLETED
*STORECIL M       1 RSTAR RSTAR COLDS
*FILES(1,FILE1,1)
*FILES(3,FILE3,1)
*CCEND


CLB, BUILD RSTAR

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  002A
*IST TABLE 3F36  0036
*PNT TABLE 3F6C  000C
*DFT TABLE 3F78  000C
MAIN RSTAR 3FAA
PNT  RSTAR 3F6E
PNT  COLDS 3F72
LIBF EBPRT 403C  3F0C
CALL UNMK  40DC
CALL PTIME 4132
LIBF MWRT  42EC  3F0F
LIBF MIOF  439D  3F12
LIBF MCOMP 4379  3F15
LIBF MDRED 4806  3F18
LIBF MDAF  47A5  3F1B
LIBF MDAI  47B2  3F1E
LIBF MDCOM 4868  3F21
LIBF MDI   47AA  3F24
CALL VIAQ  4B12
PNT  GRADE 3F76
LIBF FSUB  4B86  3F27
LIBF LDFAC 4C10  3F2A
CALL PRT   4C3E
LIBF SUBIN 4C88  3F2D
LIBF FADD  4B92  3F30
LIBF IOU   4CC2  3F33
CALL IOFIX 4D5C
CALL BT1BT 4D8C
CALL SAVE  4CF8
CALL BT2BT 4DF0
CORE       4E0E  31F2
```

(27) Continued

```
 CLB, RSTAR LD XQ

D 45 CORELOADS NOT FOUND
GRADE
DUP FUNCTION COMPLETED
```

(28)

```
// JOB         A
// *   MAINLINE CORE LOAD
// FOR COLDP
*LIST ALL
**     ON-LINE COLD START CORE LOAD
*IOCS (TYPEWRITER,KEYBOARD,DISK)
C
C      THIS IS THE SYSTEM PROCESS COLD START AND RELOAD CORE LOAD.
C      IF SENSE SWITCH 6 IS ON IT DOES A PROCESS COLD START AND IF
C      SENSE SWITCH 6 IS OFF IT ASSUMES A RELOAD CONDITION HAS OCCURED
C      SO THAT IT INITIALIZES THE SYSTEM TO THE LAST CHECK POINT.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       EXTERNAL GRADE
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 1(2,320,U,II)
       DEFINE FILE 3(3,320,U,II)
C
C      TAKE ALL CLOSED LOOPS OFF-LINE UNTIL OPERATOR BRINGS THEM ON-LINE
C
       SW0=2
C
C      UNMASK ALL LEVELS
C
       CALL UNMK(-1,-1)
C      SET UP INSKEL COMMON
       READ(1'1)RANGE,LOW,A,B
       READ(1'2)G,H
       SW5=5
       SW1=1
C
C      SET UP BASE TIMES FOR SCHEDULER
C
       CALL CLOCK(I)
       J=I/1000*1000
       K=I-J
       IBASE=J+(K/250*250)
       IBASZ=J
       IBAZZ=J/8000*8000+250
       SW3=2
       CALL COUNT(0,1,1)
C
C      TEST FOR RELOAD OR COLD START
C
       CALL SSWTCH(6,II)
       GO TO (200,100),II
  200  CONTINUE
       WRITE(1,3)
    3  FORMAT(' PROCESS COLD START')
       READ(2,2)DAY,JOBN
    2  FORMAT(I1,I5)
       CALL CHAIN(GRADE)
  100  READ(3'1)JOBN,DAY,IENDT,SW3
       READ(3'2)SETPT
       READ(3'3)AHL,ALL
       SW0=1
       GO TO (96,95),SW3
   96  CONTINUE
       CALL CLOCK(I)
       IF(I-IENDT)101,102,103
  101  IPER=IENDT-I
       GO TO 104
  102  CALL CHAIN(GRADE)
  103  IPER=(24000-IENDT)+I
  104  AA=IPER*3.6
       IF(32000.-AA)102,105,105
  105  IPER=AA
       CALL COUNT(1,2,IPER)
   95  WRITE(1,106)
```

(28) Continued

```
  106 FORMAT(' PROCESS RESTART CHECK POINT')
      CALL VIAQ
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30         AA(R )=000C          II(I )=0012           I(I )=0013
     J(I )=0014           K(I )=0015        IPER(I )=0016

STATEMENT ALLOCATIONS
 3     =0026  2     =0032  106  =0035  200  =00BB  100  =00CB  96   =00F2  101  =00FD  102  =0105  103  =0109  104  =0111
 105  =011F  95    =0129

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 GRADE    UNMK     CLOCK    COUNT    SSWTC    CHAIN    VIAQ     FSUB     FMPY     FLD      FSTO     IFIX     FLOAT    COMGO    LDFAC
 MRED     MWRT     MCOMP    MIOI     MDRED    MDCOM    MDAI     MDAF     MDI      TYPEN    HOLEB    EBPRT

REAL CONSTANTS
  .360000E 01=0018    .320000E 05=001A

INTEGER CONSTANTS
     2=001C       1=001D       5=001E    1000=001F     250=0020    8000=0021       0=0022       6=0023       3=0024   24000=0025

CORE REQUIREMENTS FOR COLDP
 COMMON       0  INSKEL COMMON    464  VARIABLES     24  PROGRAM    280


 END OF COMPILATION


 COLDP
 DUP FUNCTION COMPLETED
 *DELET    M          COLDP DUM
 DUM
 D25 NAME NOT IN L/F
 *STORECIL M       1 COLDP COLDP COLDS
 *FILES(1,FILE1,1)
 *FILES(3,FILE3,1)
 *CCEND


 CLB, BUILD COLDP

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0051
 *VTV TABLE 3F0C  002A
 *IST TABLE 3F36  0036
 *PNT TABLE 3F6C  000C
 *DFT TABLE 3F78  000C
 MAIN COLDP 3FBD
 PNT  COLDP 3F6E
 PNT  COLDS 3F72
 LIBF EBPRT 40A8  3F0C
 LIBF HOLEB 4148  3F0F
 CALL UNMK  426A
 LIBF MDRED 431C  3F12
 LIBF MDAF  42BB  3F15
 LIBF MDAI  42C8  3F18
 LIBF MDCOM 437E  3F1B
 CALL SSWTC 4628
 LIBF MWRT  47DC  3F1E
 LIBF MCOMP 4869  3F21
 LIBF MRED  47C9  3F24
 LIBF MIOI  4892  3F27
 PNT  GRADE 3F76
 LIBF MDI   42C0  3F2A
 LIBF FSUB  4CA2  3F2D
 LIBF LDFAC 4D2C  3F30
 CALL VIAQ  4D5A
 CALL PRT   4DBA
 CALL BT2BT 4E04
 CALL SAVE  4E20
 CALL IOFIX 4E84
 LIBF IOU   4EB4  3F33
 CALL BT1BT 4EEA
 CORE       4F50  30B0
```

29
```
// JOB         A
// *  MAINLINE CORE LOAD
// FOR CEINT
*LIST ALL
**    CE UNMASK CORE LOAD
*IOCS(TYPEWRITER)
C
C      THIS CORE LOAD IS FOR USE WITH THE CE INTERRUPT.  IT MAKES
C      SURE THAT ALL LEVELS ARE UNMASKED AFER USE OF THE CE INTERRUPT
C      ROUTINES.
C
       WRITE(1,1)
     1 FORMAT(' CE UNMASK CORE LOAD--PRESS START TO EXIT FROM CORE LOAD')
       PAUSE
       CALL UNMK(-1,-1)
       CALL VIAQ
       END

STATEMENT ALLOCATIONS
 1    =0006

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 UNMK     VIAQ     MWRT     MCOMP    PAUSE    TYPEN    EBPRT

INTEGER CONSTANTS
     1=0004       0=0005

CORE REQUIREMENTS FOR CEINT
 COMMON       0  INSKEL COMMON       0  VARIABLES       4  PROGRAM      54


 END OF COMPILATION


 CEINT
 DUP FUNCTION COMPLETED
 // DUP
 *STORECIL M       1 CEINT CEINT CEINT
 *CCEND

 CLB, BUILD CEINT

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0051
 *VTV TABLE 3F0C  000F
 *IST TABLE 3F1B  0036
 *PNT TABLE 3F52  0008
 MAIN CEINT 3F7E
 PNT  CEINT 3F54
 PNT  CEINT 3F58
 LIBF EBPRT 3F94  3F0C
 LIBF MWRT  41BE  3F0F
 LIBF MCOMP 424B  3F12
 LIBF PAUSE 4670  3F15
 CALL UNMK  4684
 CALL VIAQ  46CE
 CALL PRT   472E
 LIBF IOU   4778  3F18
 CALL IOFIX 4812
 CALL BT1BT 4842
 CALL SAVE  47AE
 CORE       48A8  3758

 CLB, CEINT LD XQ

DUP FUNCTION COMPLETED
```

30
```
// JOB         A
// *  THE FOLLOWING ARE THREE DUMMY CORE LOADS FOR USE IN REPLACING
// *  OR DELETING SYSTEM CORE LOADS.
```

(30) **Continued**

```
// *
// *  MAINLINE CORE LOAD
// ASM DUM

      NO ERRORS IN ABOVE ASSEMBLY.
DUM
DUP FUNCTION COMPLETED
*STORECI  M        1 DUM   DUM   DUM
*CCEND

 CLB, BUILD DUM


 CLB, DUM   LD XQ

DUP FUNCTION COMPLETED
// *  INTERRUPT CORE LOAD
// ASM IDUM

      NO ERRORS IN ABOVE ASSEMBLY.
IDUM
DUP FUNCTION COMPLETED
*STORECI  I        1 IDUM  IDUM
*CCEND

 CLB, BUILD IDUM


 CLB, IDUM  LD XQ

DUP FUNCTION COMPLETED
// *  COMBINATION CORE LOAD
// ASM CDUM

      NO ERRORS IN ABOVE ASSEMBLY.
CDUM
DUP FUNCTION COMPLETED
*STORECI  C        1 CDUM  CDUM  CDUM
*CCEND


 CLB, BUILD CDUM


 CLB, CDUM  LD XQ

DUP FUNCTION COMPLETED
```

(31)

```
// JOB        A
// *  MAINLINE CORE LOAD
// FOR GRADE
*LIST ALL
**    GRADE CHANGE PROGRAM
*IOCS(DISK,TYPEWRITER)
C
C     THIS PROGRAM STARTS THE PRODUCTION OF A NEW GRADE.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      DIMENSION INPP(8)
      DIMENSIUN RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ
      EQUIVALENCE (VALUE(2),IENDT)
      DEFINE FILE 2(100,320,U,JOBN)
      DEFINE FILE 3(3,320,U,II)
C
C     TAKE ALL LOOPS OFF CLOSED LOOP CONTROL UNTIL AFTER CHANGE
C
      SW0=2
C
C     READ NEXT SEQUENTIAL GRADE FILE OFF OF DISK
C
      IF(JOBN)999,999,9999
  999 SW3=2
      WRITE(3'1)JOBN,DAY,IENDT,SW3
      CALL PTIME(TIME)
      WRITE(1,998)TIME,DAY
      WRITE(4,998)TIME,DAY
      WRITE(5,998)TIME,DAY
```

(31) Continued

```
  998 FORMAT(' PRODUCTION STOP     TIME'F7.2,'     DAY'I2)
      CALL VIAQ
 9999 SW3=1
      IF (101-JOBN)2,2,3
    2 JOBN=1
    3 READ(2'JOBN)I,ITIME,SETPT,AHL,ALL
      DO 10 J=1,8
      IF(SETPT(J))10,10,4
    4 SETPT(J)=SETPT(J)*RANGE(J)/100.+LOW(J)
   10 CONTINUE
C
C     CALL COUNT TO SET UP TERMINATION OF GRADE ITIME SECONDS FROM NOW.
C
      CALL COUNT (1,2,ITIME)
      IENDT=ITIME/3.6
      CALL CLOCK(II)
      IENDT=IENDT+II
      IF(24000-IENDT)100,100,101
  100 IENDT=IENDT-24000
  101 CONTINUE
      WRITE(3'1)JOBN,DAY,IENDT,SW3
      WRITE(3'2)SETPT
      WRITE(3'3)AHL,ALL
      SW0=1
      II=ITIME/3600
      ITIME=(ITIME-(II*3600))/60
      TIME=II+ITIME/100.
      CALL PTIME(TIZ)
      WRITE(1,11)I,TIME,TIZ,DAY
      WRITE(4,11)I,TIME,TIZ,DAY
      WRITE(5,11)I,TIME,TIZ,DAY
   11 FORMAT(//' START OF GRADE'I6,' PRODUCTION TIME'F9.2,' START TIME'F
     19.2,' DAY'I3)
      WRITE(1,500)
  500 FORMAT(' OP-GUIDE LIMITS FOR NEW GRADE')
      WRITE(1,501)
  501 FORMAT(' POINT   HIGH LIMIT    LOW LIMIT',10X' POINT   HIGH LIMIT
     1   LOW LIMIT')
      DO 503 J=1,39,2
      K=41-J
      J1=J
      J2=J+1
  503 WRITE(1,502)J1,AHL(K),ALL(K),J2,AHL(K-1),ALL(K-1)
  502 FORMAT(I6,2F13.2,10XI6,2F13.2)
      WRITE(1,510)
  510 FORMAT(' CLOSED LOOP SET POINTS FOR NEW GRADE')
      WRITE(1,511)
  511 FORMAT(3X'POINT',5X'SETPT',5X'POINT',5X'SETPT',5X'POINT',5X'SETPT'
     1,5X'POINT',5X'SETPT')
      DO 6 J=1,8
    6 INPP(J)=(SETPT(J)-LOW(J))*100./RANGE(J)
      DO 30 I=1,5,4
      J1=I
      J2=I+1
      J3=I+2
      J4=I+3
      IA=9-I
      IB=8-I
      IC=7-I
      ID=6-I
   30 WRITE(1,7)J1,INPP(IA),J2,INPP(IB),J3,INPP(IC),J4,INPP(ID)
    7 FORMAT(I8,I10,3(I10,I10))
      CALL VIAQ
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75      IENDT(I*)=FFF6      CTIME(R )=000C      CTIZ(R )=000E
  INPP(I )=0019-0012    II(I )=001A          I(I )=001B      ITIME(I )=001C          J(I )=001D          K(I )=001E
    J1(I )=001F         J2(I )=0020         J3(I )=0021         J4(I )=0022         IA(I )=0023         IB(I )=0024
    IC(I )=0025         ID(I )=0026

STATEMENT ALLOCATIONS
 998  =003F  11   =0054  500  =0077  501  =0088  502  =00AC  510  =00B4  511  =00C9  7    =00F2  999  =0101  9999 =012E
 2    =0138  3    =013C  4    =015A  10   =017B  100  =01A0  101  =01A6  503  =022D  6    =025D  30   =02B0

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 PTIME    VIAQ     COUNT    CLOCK    FADD     FMPY     FMPYX    FDIV     FDIVX    FSTO     IFIX     FLOAT    ISTOX    STFAC    SBFAC
 MWRT     MCOMP    MIOFX    MIOIX    MIOF     MIOI     SUBSC    MDRED    MDWRT    MDCOM    MDAI     MDAF     MDI      TYPEN    EBPRT
```

31 Continued

```
REAL CONSTANTS
  .100000E 03=002C    .360000E 01=002E

INTEGER CONSTANTS
     2=0030       3=0031        1=0032       4=0033       5=0034     101=0035       8=0036  24000=0037   3600=0038      60=0039
    39=003A      41=003B        9=003C       7=003D       6=003E

CORE REQUIREMENTS FOR GRADE
 COMMON       0  INSKEL COMMON    396  VARIABLES      44  PROGRAM    696


 END OF COMPILATION


GRADE
DUP FUNCTION COMPLETED
*DELET    M          GRADE DUM
GRADE
D25 NAME NOT IN L/F
*STORECIL M       1 GRADE GRADE RSTAR
*FILES(2,FILE2,1)
*FILES(3,FILE3,1)
*CCEND

CLB, BUILD GRADE

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0036
*IST TABLE 3F42  0036
*PNT TABLE 3F78  0008
*DFT TABLE 3F80  000C
MAIN GRADE 4079
PNT  GRADE 3F7A
PNT  RSTAR 3F7E
LIBF EBPRT 4264  3F0C
LIBF MDWRT 4457  3F0F
LIBF MDI   4310  3F12
LIBF MDCOM 43CE  3F15
CALL PTIME 4684
LIBF MWRT  483E  3F18
LIBF MIOF  48EF  3F1B
LIBF MIOI  48F4  3F1E
LIBF MCOMP 48CB  3F21
CALL VIAQ  4CF0
LIBF MDRED 436C  3F24
LIBF MDAI  4318  3F27
LIBF MDAF  430B  3F2A
LIBF FADD  4D70  3F2D
LIBF STFAC 4E04  3F30
LIBF SBFAC 4E08  3F33
LIBF MIOFX 48FB  3F36
LIBF MIOIX 4900  3F39
CALL PRT   4E1C
CALL BT2BT 4E66
CALL SAVE  4E82
CALL IOFIX 4EE6
LIBF SUBIN 4F16  3F3C
LIBF IOU   4F50  3F3F
CALL BT1BT 4F86
CORE       4FEC  3014

CLB, GRADE LD XQ

DUP FUNCTION COMPLETED
```

32

```
// JOB        A
// *  TWO MINUTE LIMIT SCAN ROUTINE--COMBINATION CORE LOAD
// *  THIS PROGRAM IS INITIATED EVERY 2 MINUTES OR BY PROCESS INTERRUPT
// *
// *  FIRST PART IS THE CONVERSION AND LIMIT CHECK SUBROUTINE USED BY AI
// *  USER SUBROUTINE
// FOR
*LIST ALL
**    CONVERSION AND LIMIT CHECK SUBROUTINE
      SUBROUTINE LIMIT(I,J)
```

(32) **Continued**

```
C
C      THIS SUBROUTINE IS THE SUBROUTINE TO BE USED IN THE AIS READ AND
C      TRANSFER FUNCTION OF THE MAIN LINE CORE LOAD.
C
       INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
C
C      CONVERT I PARAMETER TO ARRAY INDEX
C
       K=40-I
       M=I+1
C
C      CONVERT AI VALUE TO ENGINEERING UNITS
C
       VSC=A(K)*LD(J)+B(K)
C
C      TEST FOR HIGH LIMIT VIOLATION
C
       IF(VSC-AHL(K))2,1,1
     1 WRITE(1,100)M,VSC
   100 FORMAT(' HIGH LIMIT VIOLATION   POINT',I3,'     VALUE',F12.4)
       GO TO 4
C
C      TEST FOR LOW LIMIT VIOLATION
C
     2 IF(VSC-ALL(K))3,3,4
     3 WRITE(1,300)M,VSC
   300 FORMAT('  LOW LIMIT VIOLATION    POINT',I3,'     VALUE',F12.4)
     4 RETURN
       END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30       CVSC(R )=0000           K(I )=0002           M(I )=0003

STATEMENT ALLOCATIONS
 100  =0008  300  =0020  1    =006C  2    =0076  3    =0082  4    =008A

FEATURES SUPPORTED
 ONE WORD INTEGERS

CALLED SUBPROGRAMS
 LD      FADDX   FSUBX   FMPYX   FLD     FSTO    FLOAT   LDFAC   MWRT    MCOMP   MIOF    MIOI    SUBSC   SUBIN

INTEGER CONSTANTS
    40=0006      1=0007

CORE REQUIREMENTS FOR LIMIT
 COMMON       0  INSKEL COMMON    464  VARIABLES      6  PROGRAM    134


 END OF COMPILATION


LIMIT
DUP FUNCTION COMPLETED
// *  LEAVE THIS SUBROUTINE IN TEMP STORAGE SINCE IT IS ONLY USED
// *  WITH THIS MAINLINE
// *  NOW COMPILE THE MAINLINE
// *  COMBINATION CORE LOAD
// FOR SCAN2
*LIST ALL
**     TWO MINUTE SCAN FOR LIMIT VIOLATIONS OF THOSE POINTS ON OP-GUIDE
*IOCS (TYPEWRITER)
C
C      THIS COMBINATION CORE LOAD SCANS ALL OF THE OP-GUIDE POINTS ON THE
C      SYSTEM AND NOTES ANY LIMIT VIOLATION TO THE OPERATOR.
C
       INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       EXTERNAL LIMIT
       DIMENSION INP(42)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
```

32 Continued

```
C
C     GET TIME IN HOURS-MINUTES
C
      CALL PTIME(TIME)
C
C     DETERMINE IF THIS IS A DEMAND SCAN OR THE NORMAL 2 MINUTE SCAN BY
C     DETERMINING WHAT LEVEL IT IS XEQ-ING ON
C
      I=7+LD(104)
      IF(LD(I)-23)1,1,2
C
C     DEMAND SCAN
C
    1 WRITE(1,3)DAY,TIME
    3 FORMAT(//,' DEMAND SCAN   DAY',I3,4X'TIME',F12.2)
      GO TO 5
C
C     NORMAL 2 MINUTE SCAN
C
    2 WRITE(1,4)DAY,TIME
    4 FORMAT(//,' NORMAL SCAN   DAY',I3,4X'TIME',F12.2)
C
C     READ 40 RELAY POINTS USING AIS READ AND TRANSFER FUNCTION
C
    5 CALL AIS(15001,INP(1),INP(42),0,LIMIT)
    6 CALL AIS(0,I)
      GO TO (6,7),I
    7 CALL DPART
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA

STATEMENT ALLOCATIONS
 3     =0036  4     =0049  1     =006D  2     =0077  5     =007F  6     =0095  7     =009F

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 LIMIT   PTIME   LD      AIS     DPART   COMGO   MWRT    MCOMP   MIOF    MIOI    TYPEN   EBPRT

INTEGER CONSTANTS
     7=0030     104=0031       23=0032       1=0033  15001=0034       0=0035

CORE REQUIREMENTS FOR SCAN2
 COMMON       0  INSKEL COMMON     464  VARIABLES      48  PROGRAM    114


 END OF COMPILATION


SCAN2
DUP FUNCTION COMPLETED
*DELET     C          SCAN2 CDUM        9999
SCAN2
D25 NAME NOT IN L/F
*STORECIL C        1 SCAN2 SCAN2 RSTAR 1515
*CCEND

CLB, BUILD SCAN2

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0021
*IST TABLE 3F2D  0036
*PNT TABLE 3F64  0008
MAIN SCAN2 3FC8
PNT  SCAN2 3F66
PNT  RSTAR 3F6A
LIBF EBPRT 400E  3F0C
CALL PTIME 40BA
CALL LD    40EA
LIBF MWRT  4284  3F0F
LIBF MIOI  433A  3F12
LIBF MIOF  4335  3F15
LIBF MCOMP 4311  3F18
CALL LIMIT 476F
CALL DPART 47C2
CALL PRT   47D0
```

(32) Continued

```
LIBF SUBIN 481A  3F1B
LIBF FADD  4874  3F1E
LIBF IOU   48F2  3F21
CALL IOFIX 498C
CALL BT1BT 49BC
CALL SAVE  4928
LIBF FADDX 486E  3F24
LIBF FSUBX 4863  3F27
LIBF LDFAC 4A20  3F2A
CALL VIAQ  4A4E
CORE       4AB0  354F

CLB, SCAN2 LD XQ

DUP FUNCTION COMPLETED
```

(33)

```
// JOB        A
// *  MAINLINE CORE LOAD
// FOR LOG15
*LIST ALL
** 15 MINUTE LOG ROUTINE
*IOCS(TYPEWRITER)
C
C      THIS IS THE FIFTEEN MINUTE LOG PROGRAM WHICH LOGS THE VALUES OF
C      ALL PROCESS VARIABLES IN THE SYSTEM
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INP(42),VSC(40),INPP(10)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
C
C      START INPUT OF RELAY POINTS
C
       CALL AIS(12001,INP(1),INP(42),0)
C
C      PRINT HEADER OF LOG
C
       CALL PTIME(TIME)
       WRITE(1,1)DAY,TIME
       WRITE(5,1)DAY,TIME
     1 FORMAT(//,' LOG15    DAY',I3,4X'TIME',F9.2)
       WRITE(1,100)
       WRITE(5,100)
   100 FORMAT(' OP-GUIDE POINTS')
       WRITE (5,2)
       WRITE (1,2)
     2 FORMAT(3X'POINT',5X'VALUE',5X'POINT',5X'VALUE',5X'POINT',5X'VALUE'
      1,5X'POINT',5X'VALUE')
C
C      START INPUT OF SS AI AND CONVERT RELAY AI WHILE SS IS COMING IN
C
       CALL AIS(02001,INPP(1),INPP(10),4096)
       DO 10 I=1,40
    10 VSC(I)=A(I)*INP(I)+B(I)
C
C      PRINT TABLE OF RELAY AI POINTS
C
       DO 20 J=1,37,4
       K=41-J
       J1=J
       J2=J+1
       J3=J+2
       J4=J+3
       WRITE(5,3)J1,VSC(K),J2,VSC(K-1),J3,VSC(K-2),J4,VSC(K-3)
    20 WRITE(1,3)J1,VSC(K),J2,VSC(K-1),J3,VSC(K-2),J4,VSC(K-3)
     3 FORMAT(4(I8,F12.2))
C
C      PRINT THE SS VALUES
C
       WRITE(1,101)
       WRITE(5,101)
   101 FORMAT(' CLOSED LOOP POINTS')
       WRITE(1,2)
       WRITE(5,2)
     4 CALL AIS(0,I)
       GO TO (4,5),I
     5 DO 6 J=1,8
```

33 Continued

```
      6 INPP(J)=(INPP(J)-LOW(J))*100./RANGE(J)
        DO 30 I=1,5,4
        J1=I
        J2=I+1
        J3=I+2
        J4=I+3
        IA=9-I
        IB=8-I
        IC=7-I
        ID=6-I
        WRITE(5,7)J1,INPP(IA),J2,INPP(IB),J3,INPP(IC),J4,INPP(ID)
     30 WRITE(1,7)J1,INPP(IA),J2,INPP(IB),J3,INPP(IC),J4,INPP(ID)
      7 FORMAT(I8,I10,3(I10,I10))
        CALL VIAQ
        END
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF       CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30       CVSC(R )=004E-0000 CTIME(R )=0050        INP(I )=007D-0054
  INPP(I )=0087-007E     I(I )=0088          J(I )=0089           K(I )=008A         J1(I )=008B          J2(I )=008C
    J3(I )=008D         J4(I )=008E         IA(I )=008F          IB(I )=0090         IC(I )=0091          ID(I )=0092

STATEMENT ALLOCATIONS
 1     =00AA  100   =00BA  2     =00C4  3     =00ED  101   =00F2  7     =00FE  10    =0154  20    =01C2  4     =01F8  5     =0202
 6     =0206  30    =0281

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 AIS      PTIME    VIAQ     FADDX    FMPY     FDIVX    FLDX     FSTO     FSTOX    IFIX     FLOAT    COMGO    ISTOX    MWRT     MCOMP
 MIOFX    MIOIX    MIOF     MIOI     SUBSC    TYPEN    EBPRT

REAL CONSTANTS
  .100000E 03=0098

INTEGER CONSTANTS
 12001=009A       0=009B       1=009C       5=009D    2001=009E    4096=009F      40=00A0      37=00A1       4=00A2      41=00A3
     2=00A4       3=00A5       8=00A6       9=00A7       7=00A8       6=00A9

CORE REQUIREMENTS FOR LOG15
 COMMON       0  INSKEL COMMON    464  VARIABLES     152  PROGRAM     530


 END OF COMPILATION


LOG15
DUP FUNCTION COMPLETED
*DELET     M          LOG15 DUM
LOG15
D25 NAME NOT IN L/F
*STORECIL M        1 LOG15 LOG15 RSTAR
*CCEND

CLB, BUILD LOG15

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0021
*IST TABLE 3F2D  0036
*PNT TABLE 3F64  0008
MAIN LOG15 4071
PNT  LOG15 3F66
PNT  RSTAR 3F6A
LIBF EBPRT 4216  3F0C
CALL PTIME 42C2
LIBF MWRT  447C  3F0F
LIBF MIOI  4532  3F12
LIBF MIOF  452D  3F15
LIBF MCOMP 4509  3F18
LIBF FADDX 4948  3F1B
LIBF MIOFX 4539  3F1E
LIBF MIOIX 453E  3F21
CALL VIAQ  49CC
CALL PRT   4A2C
LIBF SUBIN 4A76  3F24
```

(33) **Continued**

```
 LIBF FADD  494E  3F27
 LIBF IOU   4AB0  3F2A
 CALL IOFIX 4B4A
 CALL BT1BT 4B7A
 CALL SAVE  4AE6
 CORE       4BE0  3420

 CLB, LOG15 LD XQ

DUP FUNCTION COMPLETED
```

(34)

```
// JOB         A
// *  MAINLINE CORE LOAD
// FOR LOG60
** ONE HOUR LOG
*IOCS(TYPEWRITER)
*LIST ALL
C
C      THIS PROGRAM PUTS OUT THE HOUR LOG.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
C
C      PRINT HEADING
C
       CALL PTIME(TIME)
       WRITE(1,1)DAY,TIME
       WRITE(4,1)DAY,TIME
       WRITE(5,1)DAY,TIME
     1 FORMAT(//,' ONE HOUR LOG    DAY',I3,5X'TIME',F9.2)
C
C      OUTPUT ONE HOUR LOG
C
C
C      EXIT FROM ROUTINE
C
        CALL VIAQ
       END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30       CTIME(R )=0000

STATEMENT ALLOCATIONS
 1    =0005

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 PTIME    VIAQ     MWRT     MCOMP    MIOF     MIOI     TYPEN    EBPRT

INTEGER CONSTANTS
     1=0002       4=0003       5=0004

CORE REQUIREMENTS FOR LOG60
 COMMON       0  INSKEL COMMON    464  VARIABLES       2  PROGRAM     52


 END OF COMPILATION



LOG60
DUP FUNCTION COMPLETED
// DUP
*STORECIL M        1 LOG60 LOG60 RSTAR
*CCEND

CLB, BUILD LOG60

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2
```

34 Continued

```
*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0018
*IST TABLE 3F24  0036
*PNT TABLE 3F5A  0008
MAIN LOG60 3F7B
PNT  LOG60 3F5C
PNT  RSTAR 3F60
LIBF EBPRT 3F98  3F0C
CALL PTIME 4044
LIBF MWRT  41FE  3F0F
LIBF MIOI  42B4  3F12
LIBF MIOF  42AF  3F15
LIBF MCOMP 428B  3F18
CALL VIAQ  46B0
CALL PRT   4710
LIBF SUBIN 475A  3F1B
LIBF FADD  47B4  3F1E
LIBF IOU   4832  3F21
CALL IOFIX 48CC
CALL BT1BT 48FC
CALL SAVE  4868
CORE       4962  369E

CLB, LOG60 LD XQ

DUP FUNCTION COMPLETED
```

35

```
// JOB         A
// *  MAINLINE CORE LOAD
// FOR SHIFT
*LIST ALL
** SHIFT END LOG
*IOCS (TYPEWRITER)
C
C      THIS PROGRAM OUTPUTS THE SHIFT LOG
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
C
C      PRINT HEADER
C
       CALL PTIME(TIME)
       WRITE(1,1)DAY,TIME
       WRITE(4,1)DAY,TIME
       WRITE(5,1)DAY,TIME
     1 FORMAT(//,' SHIFT END LOG   DAY',I3,5X'TIME',F9.2)
C
C      OUTPUT SHIFT END LOG
C
C
C      EXIT FROM ROUTINE
C
       CALL VIAQ
       END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30      CTIME(R )=0000

STATEMENT ALLOCATIONS
 1     =0005

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 PTIME    VIAQ     MWRT     MCOMP    MIOF     MIOI     TYPEN    EBPRT

INTEGER CONSTANTS
     1=0002       4=0003       5=0004
```

(35) **Continued**

```
CORE REQUIREMENTS FOR SHIFT
 COMMON       0  INSKEL COMMON    464  VARIABLES       2  PROGRAM      52

 END OF COMPILATION


SHIFT
DUP FUNCTION COMPLETED
// DUP
*STORECIL M        1 SHIFT SHIFT RSTAR
*CCEND

CLB, BUILD SHIFT

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0018
*IST TABLE 3F24  0036
*PNT TABLE 3F5A  0008
MAIN SHIFT 3F7B
PNT  SHIFT 3F5C
PNT  RSTAR 3F60
LIBF EBPRT 3F98  3F0C
CALL PTIME 4044
LIBF MWRT  41FE  3F0F
LIBF MIOI  42B4  3F12
LIBF MIOF  42AF  3F15
LIBF MCOMP 428B  3F18
CALL VIAQ  46B0
CALL PRT   4710
LIBF SUBIN 475A  3F1B
LIBF FADD  47B4  3F1E
LIBF IOU   4832  3F21
CALL IOFIX 48CC
CALL BT1BT 48FC
CALL SAVE  4868
CORE       4962  369E

CLB, SHIFT LD XQ

DUP FUNCTION COMPLETED
```

(36)

```
// JOB         A
// *  MAINLINE CORE LOAD
// FOR WEEK
*LIST ALL
** MONDAY MORNING LOG
*IOCS(TYPEWRITER)
C
C      THIS PROGRAM OUTPUTS THE WEEKLY MONDAY MORNING LOG
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
C
C      PRINT HEADER
C
       CALL PTIME(TIME)
       WRITE(4,1)DAY,TIME
       WRITE(5,1)DAY,TIME
     1 FORMAT(//,' MONDAY MORNING REPORT    DAY',I3,5X'TIME',F9.2)
C
C      OUTPUT MONDAY MORNING REPORT
C
C
C      EXIT FROM PROGRAM
C
       CALL VIAQ
       END
```

36 Continued

```
VARIABLE ALLOCATIONS
  CSWO(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8  LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFCO OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18    A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75         G(R*)=FE72-FE54    H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30      CTIME(R )=0000

STATEMENT ALLOCATIONS
 1     =0004

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 PTIME   VIAQ    MWRT    MCOMP   MIOF    MIOI    TYPEN   EBPRT

INTEGER CONSTANTS
     4=0002       5=0003

CORE REQUIREMENTS FOR WEEK
 COMMON      0  INSKEL COMMON    464  VARIABLES      2  PROGRAM     48


 END OF COMPILATION


WEEK
DUP FUNCTION COMPLETED
// DUP
*STORECIL M        1 WEEK  WEEK  RSTAR
*CCEND

CLB, BUILD WEEK

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0018
*IST TABLE 3F24  0036
*PNT TABLE 3F5A  0008
MAIN WEEK  3F7F
PNT  WEEK  3F5C
PNT  RSTAR 3F60
LIBF EBPRT 3F94  3F0C
CALL PTIME 4040
LIBF MWRT  41FA  3F0F
LIBF MIOI  4280  3F12
LIBF MIOF  42AB  3F15
LIBF MCOMP 4287  3F18
CALL VIAQ  46AC
CALL PRT   470C
LIBF SUBIN 4756  3F1B
LIBF FADD  4780  3F1E
LIBF IOU   482E  3F21
CALL IOFIX 48C8
CALL BT1BT 48F8
CALL SAVE  4864
CORE       495E  36A2

CLB, WEEK  LD XQ

DUP FUNCTION COMPLETED
```

37

```
// JOB        A
// *  MAINLINE CORE LOAD
// FOR TREND
*LIST ALL
**    TREND LOG
*IOCS(TYPEWRITER)
C
C     THIS IS THE TREND LOG CORE LOAD.  IT READS THE VALUE THAT THE
C     OPERATOR HAS ASKED.  IT IS QUEUED PERIODICALLY BY THE TCONT
C     SUBROUTINE WITH THE PERIOD SPECIFIED BY THE OPERATOR.
C
      INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
```

37 Continued

```
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      CALL AIP(11000,INV,IPONT)
    1 CALL AIP(0,ITEST)
      GO TO (1,2),ITEST
    2 CONTINUE
      IF(4096-IPONT)4,4,100
    4 K=IPONT-4096+1
      J=9-K
      INV=(INV-LOW(J))*100./RANGE(J)
      WRITE(1,5)K,INV
    5 FORMAT(' TREND LOOP'I2,'  VALUE'I6)
      CALL VIAQ
  100 K=IPONT+1
      J=41-K
      VAL=A(J)*INV+B(J)
      WRITE(1,101)K,VAL
  101 FORMAT(' TREND OP-GUIDE'I3,'    VALUE'F10.2)
      CALL VIAQ
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18      A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54      H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30       CVAL(R )=0000         INV(I )=0002       ITEST(I )=0003
     K(I )=0004           J(I )=0005

STATEMENT ALLOCATIONS
 5     =0010  101  =001F  1     =0036  2     =0040  4     =0046  100  =0074

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 AIP      VIAQ     FADDX    FMPY     FMPYX    FDIVX    FSTO     IFIX     FLOAT    COMGO    MWRT     MCOMP    MIOF     MIOI     SUBSC
 TYPEN    EBPRT

REAL CONSTANTS
  .100000E 03=0008

INTEGER CONSTANTS
 11000=000A       0=000B   4096=000C       1=000D       9=000E      41=000F

CORE REQUIREMENTS FOR TREND
 COMMON       0  INSKEL COMMON    464  VARIABLES       8  PROGRAM     144


 END OF COMPILATION



TREND
DUP FUNCTION COMPLETED
*DELET    M          TREND DUM
TREND
D25 NAME NOT IN L/F
*STORECIL M        1 TREND TREND RSTAR
*CCEND

CLB, BUILD TREND

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0018
*IST TABLE 3F24  0036
*PNT TABLE 3F5A  0008
MAIN TREND 3F93
PNT  TREND 3F5C
PNT  RSTAR 3F60
LIBF EBPRT 3FFA  3F0C
CALL AIP   409A
LIBF MWRT  4250  3F0F
LIBF MIOI  4306  3F12
LIBF MCOMP 42DD  3F15
CALL VIAQ  4702
LIBF FADDX 477C  3F18
LIBF MIOF  4301  3F1B
CALL PRT   4800
```

(37) **Continued**

```
LIBF AIPTN 484A   3F1E
LIBF IOU   48CC   3F21
CALL IOFIX 4966
CALL BT1BT 4996
CALL SAVE  4902
CORE       49FC   3604

CLB, TREND LD XQ

DUP FUNCTION COMPLETED
```

(38)

```
// JOB         A
// *  INTERRUPT CORE LOAD
// FOR COGLP
*LIST ALL
**    CHANGE OP-GUIDE LIMITS PROGRAM
*IOCS(TYPEWRITER)
*IOCS(DISK)
C
C      THIS CORE LOAD CHANGES THE LIMITS ON OPERATOR GUIDE POINTS AT
C      OPERATOR REQUEST.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INVAL(16)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 3(3,320,U,II)
       WRITE(1,555)
   555 FORMAT(//)
       CALL CONVR(INVAL)
       WRITE (1,302)INVAL
   302 FORMAT(8I5)
       IPT=INVAL(1)*10+INVAL(9)
       AA=INVAL(2)*10000.+INVAL(3)*1000.+INVAL(4)*100.+INVAL(5)*10.+INVAL
      1(6)+INVAL(7)*.1+INVAL(8)*.01
       BB=INVAL(10)*10000.+INVAL(11)*1000.+INVAL(12)*100.+INVAL(13)*10.+I
      1NVAL(14)+INVAL(15)*.1+INVAL(16)*.01
       IF(IPT)300,300,1
     1 K=41-IPT
       IF(K)300,300,2
     2 AHL(K)=AA
       ALL(K)=BB
       WRITE(3'3)AHL,ALL
       CALL PTIME(TIME)
       WRITE(1,3)IPT,AA,BB,DAY,TIME
     3 FORMAT(' OP-GUIDE PT'I3,' HIGH LIMIT'F10.2,' LOW LIMIT'F10.2,' DAY
      1'I2,' TIME'F7.2)
       CALL INTEX
   300 WRITE (1,301)
   301 FORMAT(' INVALID ENTRY OP-REQUEST P1')
       CALL INTEX
       END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30         AA(R )=0006         BB(R )=0008       CTIME(R )=000A
 INVAL(I )=0027-0018     II(I )=0028         IPT(I )=0029          K(I )=002A

STATEMENT ALLOCATIONS
 555  =004A  302  =004D  3    =0050  301  =0071  1    =0132  2    =013C  300  =0168

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 CONVR   PTIME   INTEX   FADD    FMPY    FLD     FSTO    FSTOX   FLOAT   MWRT    MCOMP   MIOAI   MIOF    MIOI    SUBSC
 MDWRT   MDCOM   MDAF    TYPEN   EBPRT

REAL CONSTANTS
  .100000E 05=003A   .100000E 04=003C   .100000E 03=003E   .100000E 02=0040   .100000E 00=0042   .100000E-01=0044

INTEGER CONSTANTS
      1=0046      10=0047      41=0048       3=0049
```

(38) Continued

```
CORE REQUIREMENTS FOR COGLP
 COMMON       0  INSKEL COMMON    464  VARIABLES      58  PROGRAM    308


 END OF COMPILATION



COGLP
DUP FUNCTION COMPLETED
*DELET    I          COGLP IDUM         9999
COGLP
D25 NAME NOT  IN L/F
*STORECIL I        1 COGLP COGLP COGLP 1100
*FILES(3,FILE3,1)
*CCEND

CLB, BUILD COGLP

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0027
*PNT TABLE 3F34  0004
*DFT TABLE 3F38  0006
MAIN COGLP 3FB9
PNT  COGLP 3F36
LIBF EBPRT 40A6  3F0C
LIBF MWRT  42D0  3F0F
LIBF MCOMP 435D  3F12
CALL CONVR 4788
LIBF MIOAI 43A7  3F15
LIBF FADD  47DC  3F18
LIBF MDWRT 49AD  3F1B
LIBF MDAF  4861  3F1E
LIBF MDCOM 4924  3F21
CALL PTIME 4BDA
LIBF MIOI  4386  3F24
LIBF MIOF  4381  3F27
CALL PRT   4C0A
LIBF IOU   4C54  3F2A
CALL IOFIX 4CEE
CALL BT1BT 4D1E
CALL SAVE  4C8A
LIBF SUBIN 4D82  3F2D
CALL GETVL 4DE6
LIBF FADDX 47D6  3F30
CALL BT2BT 4E82
CORE       4EA0  315F

CLB, COGLP LD XQ

DUP FUNCTION COMPLETED
```

(39)

```
// JOB         A
// *  INTERRUPT CORE LOAD
// FOR CCLSP
*LIST ALL
**    CHANGE OF CLOSED LOOP CONTROL SET POINT
*IOCS(TYPEWRITER)

*IOCS(DISK)
C
C      THIS CORE LOAD CHANGES THE SET POINT VALUE FOR A SET POINT
C      STATION UPON OPERATOR REQUEST.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INVAL(16)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 3(3,320,U,II)
       WRITE(1,555)
```

39 Continued

```
   555 FORMAT(//)
       CALL CONVR(INVAL)
       WRITE(1,100)INVAL
   100 FORMAT(8I5)
       CALL PTIME(TIME)
       I=INVAL(1)
       J=INVAL(7)*10+INVAL(8)
       IF(I)300,300,1
     1 K=9-I
       IF(K)300,300,2
     2 IF(J)300,300,3
     3 SETPT(K)=J*RANGE(K)/100.+LOW(K)
       WRITE(3'2)SETPT
       WRITE(1,4)I,J,DAY,TIME
     4 FORMAT(' LOOP'I3,' NEW SET POINT'I4,' DAY'I2,' TIME'F7.2)
       CALL INTEX
   300 WRITE(1,301)
   301 FORMAT(' INVALID ENTRY OP-REQUEST P2')
       CALL INTEX
       END
 VARIABLE ALLOCATIONS
   CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
    DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
  COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
  IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
  IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30       CTIME(R )=0006       INVAL(I )=0019-000A     II(I )=001A
      I(I )=001B           J(I )=001C           K(I )=001D

 STATEMENT ALLOCATIONS
  555  =0029  100  =002C  4    =002F  301  =0047  1    =007F  2    =0089  3    =008D  300  =00BF

 FEATURES SUPPORTED
  ONE WORD INTEGERS
  IOCS

 CALLED SUBPROGRAMS
  CONVR    PTIME    INTEX    FADD     FMPYX    FDIV     FSTO     IFIX     FLOAT    ISTOX    MWRT     MCOMP    MIOAI    MIOF     MIOI
  SUBSC    MDWRT    MDCOM    MDAI     TYPEN    EBPRT

 REAL CONSTANTS
   .100000E 03=0022

 INTEGER CONSTANTS
       1=0024      10=0025       9=0026       3=0027       2=0028

 CORE REQUIREMENTS FOR CCLSP
  COMMON       0  INSKEL COMMON    464  VARIABLES     34  PROGRAM    164


  END OF COMPILATION



 CCLSP
 DUP FUNCTION COMPLETED
 *DELET    I          CCLSP IDUM         9999
 CCLSP
 D25 NAME NOT IN L/F
 *STORECIL I       1 CCLSP CCLSP CCLSP 1101
 *FILES(3,FILE3,1)
 *CCEND

 CLB, BUILD CCLSP

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0051
 *VTV TABLE 3F0C  0027
 *PNT TABLE 3F34  0004
 *DFT TABLE 3F38  0006
 MAIN CCLSP 3F8F
 PNT  CCLSP 3F36
 LIBF EBPRT 3FFE  3F0C
 LIBF MWRT  4228  3F0F
 LIBF MCOMP 42B5  3F12
 CALL CONVR 46E0
 LIBF MIOAI 42FF  3F15
 CALL PTIME 4720
 LIBF FADD  4770  3F18
 LIBF MDWRT 4941  3F1B
 LIBF MDAI  4802  3F1E
 LIBF MDCOM 48B8  3F21
```

39 **Continued**

```
LIBF MIOI   42DE   3F24
LIBF MIOF   42D9   3F27
CALL PRT    4B62
LIBF IOU    4BAC   3F2A
CALL IOFIX 4C46
CALL BT1BT 4C76
CALL SAVE   4BE2
LIBF SUBIN 4CDA   3F2D
CALL GETVL 4D3E
LIBF FADDX 476A   3F30
CALL BT2BT 4DDA
CORE        4DF8   3207

CLB, CCLSP LD XQ

DUP FUNCTION COMPLETED
```

40

```
// JOB          A
// *   INTERRUPT CORE LOAD
// FOR MGRTP
*LIST ALL
**     MODIFY GRADE RUN TIME PROGRAM
*IOCS(TYPEWRITER)
*IOCS(DISK)
C
C      THIS CORE LOAD CHANGES THE RUN TIME FOR A GRADE UPON OPERATOR
C      REQUEST.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INVAL(16)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 3(3,320,U,II)
       WRITE(1,555)
  555 FORMAT(//)
       CALL CONVR(INVAL)
       WRITE(1,302)INVAL
  302 FORMAT(8I5)
       IH=INVAL(5)*10+INVAL(6)
       IM=INVAL(7)*10+INVAL(8)
       IF(24-IH)300,300,20
   20 IF(60-IM)300,300,21
   21 CALL CLOCK(I)
       K=IH*1000+IM*100/6
       IF(K-I)1,2,2
    1 IPER=24000-K+I
       GO TO 3
    2 IPER=K-I
    3 AA=IPER*3.6
       IF(32000.-AA)4,5,5
    4 WRITE(1,40)
   40 FORMAT(' TOO LONG OF A RUN TIME')
       CALL INTEX
    5 IPER=AA
       IENDT=K
       WRITE(3'1)JOBN,DAY,IENDT,SW3
       TIME=(IH*100+IM)/100.
       WRITE(1,6)TIME
    6 FORMAT(' JOB WILL NOW TERMINATE AT'F7.2)
       CALL COUNT(1,2,IPER)
       CALL INTEX
  300 WRITE(1,301)
  301 FORMAT(' INVALID ENTRY OP-REQUEST P3')
       CALL INTEX
       END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA

STATEMENT ALLOCATIONS
 555   =0036  302   =0039  40    =003C  6     =004A  301   =005A  20    =0098  21    =009E  1     =00BA  2     =00C4  3     =00CA
 4     =00D8  5     =00DE  300   =010C

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 CONVR    CLOCK    INTEX    COUNT    FSUB     FMPY     FDIV     FLD      FSTO     IFIX     FLOAT    LDFAC    MWRT     MCOMP    MIOAI
 MIOF     MDWRT    MDCOM    MDI      TYPEN    EBPRT
```

40 Continued

```
REAL CONSTANTS
  .360000E 01=0026    .320000E 05=0028     .100000E 03=002A

INTEGER CONSTANTS
     1=002C      10=002D      24=002E      60=002F    1000=0030     100=0031       6=0032  24000=0033       3=0034       2=0035

CORE REQUIREMENTS FOR MGRTP
 COMMON      0  INSKEL COMMON    464  VARIABLES     38  PROGRAM    236


 END OF COMPILATION



MGRTP
DUP FUNCTION COMPLETED
*STORECIL I         1 MGRTP MGRTP MGRTP 1102
*FILES(3,FILE3,1)
*CCEND

CLB, BUILD MGRTP

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0027
*PNT TABLE 3F34  0004
*DFT TABLE 3F38  0006
MAIN MGRTP 3FA2
PNT  MGRTP 3F36
LIBF EBPRT 404A  3F0C
LIBF MWRT  4274  3F0F
LIBF MCOMP 4301  3F12
CALL CONVR 472C
LIBF MIOAI 434B  3F15
LIBF FSUB  4774  3F18
LIBF LDFAC 47FE  3F1B
LIBF MDWRT 497F  3F1E
LIBF MDI   4838  3F21
LIBF MDCOM 48F6  3F24
LIBF MIOF  4325  3F27
CALL PRT   4BA0
LIBF IOU   4BEA  3F2A
CALL IOFIX 4C84
CALL BT1BT 4CB4
CALL SAVE  4C20
LIBF SUBIN 4D18  3F2D
CALL GETVL 4D7C
LIBF FADDX 477A  3F30
CALL BT2BT 4E18
CORE       4E36  31C9

CLB, MGRTP LD XQ

DUP FUNCTION COMPLETED
```

41

```
// JOB         A
// *  INTERRUPT CORE LOAD
// FOR CPJSP
*LIST ALL
**    CHANGE PROCESS JOB SEQUENCE
*IOCS(TYPEWRITER)
*IOCS(DISK)
C
C      THIS CORE LOAD CHANGES THE SEQUENCE OF GRADES UPON OPERATOR
C      REQUEST.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INVAL(16)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 3(3,320,U,II)
       WRITE(1,555)
```

41 Continued

```
  555 FORMAT(//)
      CALL CONVR(INVAL)
      WRITE(1,1)INVAL
    1 FORMAT(8I5)
      I=INVAL(7)*10+INVAL(8)
      AA=INVAL(4)*10000.+INVAL(5)*1000.+INVAL(6)*100.+I
      IF(AA)4,4,22
   22 IF(32000.-AA)300,300,2
    2 IF(I)3,3,4
    3 I=100
    4 JOBN=I
      WRITE(3'1)JOBN,DAY,IENDT,SW3
      CALL PTIME(TIME)
      I=AA
      WRITE(1,5)I,JOBN,DAY,TIME
    5 FORMAT(' NEXT JOB'I6,'  QUEUE SEQUENCE'I4,'  DAY'I2,'  TIME'F7.2)
      CALL INTEX
  300 WRITE(1,301)
  301 FORMAT(' INVALID ENTRY OP-REQUEST P4')
      CALL INTEX
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30         AA(R )=0006       CTIME(R )=0008       INVAL(I )=001F-0010
    II(I )=0020          I(I )=0021

STATEMENT ALLOCATIONS
 555  =0034  1    =0037  5    =003A  301  =0056  22   =00AC  2    =00B3  3    =00B7  4    =00BB  300  =00E1

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 CONVR    PTIME    INTEX    FADD     FSUB     FMPY     FLD      FSTO     IFIX     FLOAT    LDFAC    MWRT     MCOMP    MIOAI    MIOF
 MIOI     MDWRT    MDCOM    MDI      TYPEN    EBPRT

REAL CONSTANTS
  .100000E 05=0028    .100000E 04=002A    .100000E 03=002C    .320000E 05=002E

INTEGER CONSTANTS
     1=0030      10=0031     100=0032       3=0033

CORE REQUIREMENTS FOR CPJSP
 COMMON       0  INSKEL COMMON    464  VARIABLES     40  PROGRAM    192


 END OF COMPILATION


CPJSP
DUP FUNCTION COMPLETED
*DELET    I          CPJSP IDUM        9999
CPJSP
D25 NAME NOT IN L/F
*STORECIL I       1 CPJSP CPJSP CPJSP 1103
*FILES(3,FILE3,1)
*CCEND

CLB, BUILD CPJSP

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  002D
*PNT TABLE 3F3A  0004
*DFT TABLE 3F3E  0006
MAIN CPJSP 3FA4
PNT  CPJSP 3F3C
LIBF EBPRT 4026  3F0C
LIBF MWRT  4250  3F0F
LIBF MCOMP 42DD  3F12
CALL CONVR 4708
LIBF MIOAI 4327  3F15
LIBF FADD  475C  3F18
LIBF LDFAC 47DA  3F1B
LIBF FSUB  4750  3F1E
```

(41) **Continued**

```
LIBF MDWRT 495B  3F21
LIBF MDI   4814  3F24
LIBF MDCOM 48D2  3F27
CALL PTIME 4B88
LIBF MIOI  4306  3F2A
LIBF MIOF  4301  3F2D
CALL PRT   4BB8
LIBF IOU   4C02  3F30
CALL IOFIX 4C9C
CALL BT1BT 4CCC
CALL SAVE  4C38
LIBF SUBIN 4D30  3F33
CALL GETVL 4D94
LIBF FADDX 4756  3F36
CALL BT2BT 4E30
CORE       4E4E  31B1

CLB, CPJSP LD XQ

DUP FUNCTION COMPLETED
```

(42)

```
// JOB         A
// *  INTERRUPT CORE LOAD
// FOR STRND
*LIST ALL
**    START TREND LOG ROUTINE
*IOCS(TYPEWRITER)
C
C     THIS CORE LOAD INITIATES A TREND LOG OF THE POINT SPECIFIED BY
C     THE OPERATOR.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      DIMENSION INVAL(16)
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      WRITE(1,555)
  555 FORMAT(//)
      CALL CONVR(INVAL)
      WRITE(1,1)INVAL
    1 FORMAT(8I5)
      II=INVAL(6)*100+INVAL(7)*10+INVAL(8)
      JJ=INVAL(14)*100+INVAL(15)*10+INVAL(16)
      IF(INVAL(1))300,2,100
    2 I=INVAL(2)*10+INVAL(3)
      IF(I)300,300,3
    3 J=I-1
      IF(40-J)300,300,4
    4 IPONT=J
      WRITE(1,5)I,II,JJ
    5 FORMAT (' TREND LOG OP-GUIDE POINT'I3,'  PERIOD'I5,'    COUNT'I5)
      GO TO 200
  100 IF(INVAL(1)-1)300,101,300
  101 IF(INVAL(3))300,300,102
  102 IF(INVAL(3)-8)103,103,300
  103 IPONT =4095+INVAL(3)
      WRITE(1,104)INVAL(3),II,JJ
  104 FORMAT(' TREND LOG LOOP'I2,'  PERIOD'I6,'    COUNT'I6)
  200 IPERD=II
      ITCNT=JJ
      CALL COUNT(2,3,2)
      CALL INTEX
  300 WRITE(1,301)
  301 FORMAT(' INVALID ENTRY OP-REQUEST P5')
      CALL INTEX
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30      INVAL(I )=0011-0002    II(I )=0012          JJ(I )=0013

 STATEMENT ALLOCATIONS
 555  =0026  1    =0029  5    =002C  104  =0049  301  =0061  2    =00B7  3    =00C8  4    =00D4  100  =00E4  101  =00EC
 102  =00F2  103  =00FA  200  =010E  300  =011D
```

42 **Continued**

```
FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 CONVR   COUNT   INTEX   MWRT    MCOMP   MIOAI   MIOIX   MIOI    TYPEN   EBPRT

INTEGER CONSTANTS
     1=001E    100=001F     10=0020     40=0021      8=0022   4095=0023      2=0024      3=0025

CORE REQUIREMENTS FOR STRND
 COMMON       0  INSKEL COMMON    464  VARIABLES     30  PROGRAM    262


 END OF COMPILATION


STRND
DUP FUNCTION COMPLETED
*DELET    I         STRND IDUM        9999
STRND
D25 NAME NOT IN L/F
*STORECIL I       1 STRND STRND       1104
*CCEND

CLB, BUILD STRND

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  001B
*PNT TABLE 3F28  0004
MAIN STRND 3F9D
PNT  STRND 3F2A
LIBF EBPRT 4050  3F0C
LIBF MWRT  427A  3F0F
LIBF MCOMP 4307  3F12
CALL CONVR 4732
LIBF MIOAI 4351  3F15
LIBF MIOI  4330  3F18
LIBF MIOIX 433C  3F1B
CALL PRT   4766
LIBF IOU   47B0  3F1E
CALL IOFIX 484A
CALL BT1BT 487A
CALL SAVE  47E6
LIBF SUBIN 48DE  3F21
CALL GETVL 4942
LIBF FADDX 49F8  3F24
CORE       4A7E  3581

CLB, STRND LD XQ

DUP FUNCTION COMPLETED
```

43

```
// JOB         A
// FOR AIMON
*LIST ALL
**     ANALOG INPUT LOG ROUTINE
*IOCS(TYPEWRITER)
C
C      THIS CORE LOAD LOGS ALL INFORMATION ABOUT ANY ANALOG INPUT POINT
C      ON THE SYSTEM UPON OPERATOR REQUEST.
C
C
C      POINTS ARE SELECTED AS FOLLOWS --
C      OP-GUIDE POINTS - D1=0, D7-8=01-40
C      CLOSED-LOOP POINTS - D1=1, D7-8=01-08
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION INVAL(16)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
```

(43) Continued

```
      WRITE(1,555)
  555 FORMAT(//)
C
C     READ DATA ENTRY DIALS
C
      CALL CONVR(INVAL)
      WRITE(1,900)INVAL
  900 FORMAT (8I5)
C
C     CONVERT POINT DESIGNATION, D7-8
C
      IX1 = INVAL(7)*10 + INVAL(8)
C
C     MUST BE GREATER THAN 0
C
      IF(IX1)800,800,10
C
C     CHECK D1
C
   10 IF(INVAL(1)-1)20,500,800
C
C     CHECK INPUT OP-GUIDE POINT DESIGNATION
C
   20 IF(IX1-40)30,30,800
C
C     READ OP-GUIDE POINT
C
   30 IX2 = IX1-1
      CALL AIP (11000,VALUE(1),IX2)
   35 CALL AIP (0,IX2)
      GO TO (35,40), IX2
C
C     CONVERT VALUE SELECTED
C
   40 IX2 = 41-IX1
      VAL = A(IX2)*VALUE(1) + B(IX2)
C
C     PRINT TIME AND HEADER
C
      CALL PTIME(TIME)
      WRITE (1,920)DAY,TIME
  920 FORMAT (/' DAY'I2,' TIME'F7.2)
      WRITE (1,930)
  930 FORMAT  (' OP-GUIDE POINT   VALUE   HIGH LIMIT   LOW LIMIT     FAC
     -TOR A        FACTOR B')
      WRITE (1,940)IX1,VAL,AHL(IX2),ALL(IX2),A(IX2),B(IX2)
      CALL INTEX
  940 FORMAT (7X,I2,4X,F10.2,1X,F10.2,2X,F10.2,5X,E13.6,3X,E13.6)
C
C     CLOSED-LOOP POINT, CHECK LIMIT ON DESIGNATION
C
  500 IF(IX1-8)510,510,800
  510 IX2 = IX1+4095
      CALL AIP (01000,VALUE(1),IX2)
  520 CALL AIP (0,IX2)
      GO TO (520,530),IX2
C
C     CONVERT SETPT FOR OUTPUT
C
  530 IX2 = 9-IX1
      VALUE(1) = (VALUE(1)-LOW(IX2))*100./RANGE(IX2)
      IAL = (SETPT(IX2)-LOW(IX2))*100./RANGE (IX2)
C
C     PRINT DAY, TIME, AND HEADER
C
      CALL PTIME(TIME)
      WRITE (1,920)DAY,TIME
      WRITE (1,950)
  950 FORMAT  (' CLOSED-LOOP POINT   VALUE      SETPT        OFFLN
     -  RANGE          LOW')
C
C     WRITE VALUES
C
      WRITE (1,960)IX1,VALUE(1),IAL,OFFLN(IX2),RANGE(IX2),LOW(IX2)
      CALL INTEX
  960 FORMAT (8X,I2,7X,I8,5X,I6,10X,I2,7X,F10.2,4X,I6)
  800 WRITE (1,810)
      CALL INTEX
  810 FORMAT(' INVALID ENTRY OP-REQUEST P6')
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54      H(R*)=FE52-FE34 IENDT(I*)=FE33
```

(43) Continued

```
STATEMENT ALLOCATIONS
 555  =0028  900  =002B  920  =002E  930  =0039  940  =0062  950  =006F  960  =0097  810  =00A4  10   =00D3  20   =00DD
 30   =00E3  35   =00F5  40   =00FF  500  =013E  510  =0144  520  =0156  530  =0160  800  =01BB

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 CONVR   AIP     PTIME   INTEX   FADDX   FMPY    FDIVX   FLDX    FSTO    IFIX    FLOAT   COMGO   ISTOX   MWRT    MCOMP
 MIOAI   MIOFX   MIOIX   MIOF    MIOI    SUBSC   TYPEN   EBPRT

REAL CONSTANTS
  .100000E 03=001C

INTEGER CONSTANTS
      1=001E     10=001F     40=0020  11000=0021      0=0022     41=0023      8=0024   4095=0025   1000=0026      9=0027

CORE REQUIREMENTS FOR AIMON
 COMMON      0  INSKEL COMMON    464  VARIABLES     28  PROGRAM    422


 END OF COMPILATION



AIMON
DUP FUNCTION COMPLETED
*DELET    I          AIMON IDUM         9999
AIMON
D25 NAME NOT IN L/F
*STORECIL I        1 AIMON AIMON        1105
*CCEND

CLB, BUILD AIMON

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0027
*PNT TABLE 3F34  0004
MAIN AIMON 3FEC
PNT  AIMON 3F36
LIBF EBPRT 40FA  3F0C
LIBF MWRT  4324  3F0F
LIBF MCOMP 43B1  3F12
CALL CONVR 47DC
LIBF MIOAI 43FB  3F15
CALL AIP   4810
LIBF FADDX 4856  3F18
CALL PTIME 48E6
LIBF MIOI  43DA  3F1B
LIBF MIOF  43D5  3F1E
LIBF MIOFX 43E1  3F21
LIBF MIOIX 43E6  3F24
CALL PRT   4916
LIBF IOU   4960  3F27
CALL IOFIX 49FA
CALL BT1BT 4A2A
CALL SAVE  4996
LIBF SUBIN 4A8E  3F2A
CALL GETVL 4AF2
LIBF AIPTN 4B8E  3F2D
LIBF FADD  485C  3F30
CORE       4C12  33ED

CLB, AIMON LD XQ

DUP FUNCTION COMPLETED
```

44

```
// JOB         A
// DUP
*DUMPLET

        LET

 PACK LABEL
  00000

.DCOM 0010   0000       .MBT  0020   0010       .SKSB 0020   0030       .SUP  00B0   0050       .CLB  00A0   0100       .DUP  0440   01A0
.ASM  0300   05E0       .FOR  0680   08E0       .SIM  05F0   0F60       .LET  0080   1550       IAND  0002   15D0       CLEAR 0009   15D2
CLOCK 0002   15DB       COUNT 0004   15DD       DMP   0017   15E1       DMPHX                    DMPDC                    DMPS  0010   15F8
DMPST                   DPART 0002   1608       ENDTS 0002   160A       IEOR  0002   160C       LD    0002   160E       LEVEL 0004   1610
MASK  0003   1614       OPMON 0002   1617       IOR   0002   1619       QIFON 000A   161B       QUEUE 000C   1625       RESMK 0004   1631
SAVMK 0003   1635       SETCL 0003   1638       TIMER 0006   163B       UNMK  0005   1641       UNQ   0005   1646       VIAQ  0007   164B
CONHX 0006   1652       TRPRT 0007   1658       FLIP  0007   165F       EADD  000B   1666       ESUB                     EADDX
ESUBX                   ESBR                     ESBRX                    EATN  000D   1671       EATAN                    EAVL  0003   167E
EABS                    EAXB  0006   1681       EAXBX                    EAXI  0006   1687       EAXIX                    EDVR  0007   168D
EDVRX                   EDIV                     EDIVX                    ELD   0009   1694       ELDX                     ESTO
ESTOX                   ELN   000B   169D       EALOG                    EMPY  0004   16A8       EMPYX                    ESINE 000D   16AC
ESIN                    ECOSN                    ECOS                     ESQR  0007   16B9       ESQRT                    ETNH  0006   16C0
ETANH                   ETRTN 0004   16C6       ETNTR                    EXPN  000B   16CA       EEXP                     FSBR  000B   16D5
FSBRX                   FADD                     FSUB                     FADDX                    FSUBX                    FARC  0004   16E0
FATN  000C   16E4       FATAN                    FAVL  0003   16F0       FABS                     FAXB  0006   16F3       FAXBX
FAXI  0006   16F9       FAXIX                    FBTD  001A   16FF       FDTB                     FDIV  0008   1719       FDIVX
FDVR                    FDVRX                    FIXIX 0005   1721       FIXI                     FLD   0009   1726       FLDX
FSTO                    FSTOX                    FLN   000B   172F       FALOG                    FLOAT 0003   173A       FMPY  0005   173D
FMPYX                   FSINE 000B   1742       FSIN                     FCOSN                    FCOS                     FSQR  0007   174D
FSQRT                   FTNH  0006   1754       FTANH                    FTRTN 0004   175A       FTNTR                    FXPN  0009   175E
FEXP                    IABS  0003   1767       IFIX  0004   176A       NORM  0004   176E       SNR   0003   1772       XDD   0006   1775
XMD   0005   177B       XMDS  0004   1780       XSQR  0004   1784       BINDC 0006   1788       BINHX 0004   178E       DCBIN 0006   1792
EBPA  0006   1798       EBPRT 000A   179E       HOLEB 0012   17A8       HOLPR 000D   17BA       HXBIN 0005   17C7       PAPEB 0010   17CC
PAPHL 0014   17DC       PAPPR 0011   17F0       PRT   0005   1801       ADRCK 0007   1806       COMGO 0006   180D       COMG1
DATSW 0004   1813       DVCHK 0002   1817       ESIGN 0005   1819       FCTST 0003   181E       FSIGN 0005   1821       IOU   0007   1826
ISIGN 0003   182D       ISTOX 0003   1830       LDFAC 0004   1833       STFAC                    SBFAC                    DVFAC
MDFIO 0023   1837       MDAF                     MDAI                     MDCOM                    MDF                      MDFX
MDI                     MDIX                     MDRED                    MDWRT                    MDFND 0008   185A       MFIO  0059   1862
MRED                    MWRT                     MCOMP                    MIOAF                    MIOAI                    MIOFX
MIOIX                   MIOF                     MIOI                     MGOTO 000E   18BB       MFIF                     MIIF
MEIF                    MIAR  000E   18C9       MIARX                    MFAR                     MFARX                    MEAR
MEARX                   OVERF 0002   18D7       PAUSE 0002   18D9       REWND 0009   18DB       BCKSP                    EOF
SAVE  000A   18E4       IOFIX                    SLITE 0006   18EE       SLITT                    SSWTC 0004   18F4       STOP  0003   18F8
SUBIN 0005   18FB       SUBSC 0004   1900       TSTOP 0002   1904       TSTRT 0002   1906       TTEST 0003   1908       TSET
UFIO  001C   190B       URED                     UWRT                     UIOI                     UIOF                     UIOAI
UIOAF                   UIOFX                    UIOIX                    UCOMP                    PLOTX 000D   1927       CARDN 0016   1934
PAPTN 0010   194A       MAGT  0020   195A       AIPTN 0009   197A       AIPN                     AISQN 000F   1983       AISN
AIRN  000D   1992       ANINT 0014   199F       DINP  0013   19B3       DIEXP 0006   19C6       DICMP 0007   19CC       DAOP  0013   19D3
IOPE  0009   19E6       OUSLY                    ETS                      XSAVE 0009   19EF       XEXIT                    XLOAD
GAGED 0003   19F8       UNGAG                    AIP   0004   19FB       AIS   000D   19FF       AIR   0011   1A0C       CS    0008   1A1D
VS                      DI                       PI                       CSC   000A   1A25       VSC                      DIC
PIC                     CSX   0004   1A2F       VSX                      DIX                      PIX                      DAC   0007   1A33
CO                      DO                       PO                       QZERQ 0002   1A3A       QZ010 0006   1A3C       BT1BT 0007   1A42
BT2BT 0003   1A49       FCHAR 0005   1A4C       SCALF 0002   1A51       FGRID 0007   1A53       FPLOT 0004   1A5A       ECHAR 0005   1A5E
SCALE 0002   1A63       EGRID 0008   1A65       EPLOT 0005   1A6D       POINT 0007   1A72       FCHRX 0024   1A79       FCHRI
WCHRI                   FRULE 0009   1A9D       FMOVE                    FINC                     ECHRX 0025   1AA6       ECHRI
VCHRI                   ERULE 000B   1ACB       EMOVE                    EINC                     XYPLT 0007   1AD6       PLOTI 0003   1ADD
PLOTS                   .TEMP 1AE0   1B00       .E    5A00   1B00



         FLET

PACK LABEL
 00000

9DUMY 00A0   05A0       .E    00A0   05A0



        LET

 PACK LABEL
  11111

.LET  0080   0000       SYDIR 009E   0080       OUTTR                    CHAIN                    INTEX                    SHARE
SPECL                   BACK                     EACLK                    SCHED 0014   011E       LEV10 0024   0132       SOUT  0003   0156
QUE15 0002   0159       TCONT 0003   015B       TABRT 0002   015E       GETVL 000B   0160       CONVR 0005   016B       PTIME 0005   0170
IADDR 0002   0175       ISBAD 0002   0177       CESET 0002   0179       ABORT 0002   017B       ENDGD 0002   017D       .TEMP 017F   0180
.E    1180   0180
```

44 Continued

```
        FLET

PACK LABEL
 11111

.PRWS 0051  1118    .FIOS 000F  1169    .MESS 00A3  1178    /EPDM 7FFF  121B    /EPSV 0780  1282    /INSV 48FF  1288
/NPSV 4180  12C3    FILE1 0002  12F8    FILE2 0064  12FA    FILE3 0003  135E    COLDN 05DC  1361    COLDS 00D6  1366
RSTAR 0F8C  1367    COLDP 10CE  1374    CEINT 0A26  1382    DUM   012C  138B    IDUM  0094  138C    CDUM  013A  938D
GRADE 116A  138E    SCAN2 0C2E  939C    LOG15 0D5E  13A6    LOG60 0AE0  13B1    SHIFT 0AE0  13BA    WEEK  0ADC  13C3
TREND 0B7A  13CC    COGLP 101E  13D6    CCLSP 0F76  13E3    MGRTP 0FB4  13F0    CPJSP 0FCC  13FD    STRND 0BFC  140A
AIMON 0D90  1414    9DUMY 0159  141F    /SPSV 4180  1578    /PRSV 4180  15AD    .SKEL 0036  15E2    .EPRG 0022  1618
/CLST 0780  163A    .E    0280  12F8

DUP FUNCTION COMPLETED


// JOB         A
// END OF ALL JOBS
```

45

```
// JOB         A
// *  NONPROCESS CORE LOAD
// *  SPECIAL JOB TO SET UP FILES ON DISK
// FOR SPECL
*LIST ALL
**    SPECIAL PROGRAM TO SET UP FILES ON DISK
*IOCS(DISK,1443 PRINTER)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
C     THIS IS A SPECIAL ONE TIME ONLY CORE LOAD TO SET UP THE JOB
C     FILES ON DISK FOR TEST PURPOSES.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      DEFINE FILE 1(2,320,U,II)
      DEFINE FILE 2(100,320,U,JOBN)
      DO 10 J=1,40
      AHL(J)=32000.
      ALL(J)=-32000.
      A(J)=-1.
   10 B(J)=0.
      DO 11 J=1,8
      RANGE(J)=-32000.
   11 LOW(J)=0
      WRITE(1'1)RANGE,LOW,A,B
      DO 12 J=4,11
      K=40-J
      AHL(K)=5.5
   12 ALL(K)=4.5
      KK=0
      K=20
      ITIME=20*60
      IX=-1
      DO 13 J=1,8
   13 SETPT(J)=0
      JOBN=1
      DO 100 J=1,100
      SETPT(1)=50+IX*20
      SETPT(2)=50-IX*20
      IX=IX*(-1)
      WRITE(2'JOBN)J,ITIME,SETPT,AHL,ALL
      WRITE(3,14)J,KK,K
   14 FORMAT(8I5)
      WRITE(3,14)SETPT(8),SETPT(7),SETPT(6),SETPT(5),SETPT(4),SETPT(3),S
     1ETPT(2),SETPT(1)
      DO 20 IB=1,39,2
      IC=41-IB
      ID=IB+1
   20 CONTINUE
   15 FORMAT(2(I10,2F10.2))
  100 CONTINUE
      CALL EXIT
      END
```

(45) Continued

```
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75         G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30        II(I )=000C          J(I )=000D          K(I )=000E
    KK(I )=000F      ITIME(I )=0010         IX(I )=0011        IB(I )=0012         IC(I )=0013         ID(I )=0014

UNREFERENCED STATEMENTS
 15

STATEMENT ALLOCATIONS
 14    =0036  15    =0039  10    =005A  11    =0077  12    =00AC  13    =00D3  20    =015C  100   =0166

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 FLD      FSTOX    ISTOX    STFAC    SBFAC    MWRT     MCOMP    MIOIX    MIOI     SUBSC    SNR      MDWRT    MDCOM    MDAI     MDAF
 MDI      PRNTN    EBPRT

REAL CONSTANTS
  .320000E 05=001E    .100000E 01=0020    .000000E 00=0022    .550000E 01=0024    .450000E 01=0026

INTEGER CONSTANTS
     1=0028      40=0029       8=002A       0=002B       4=002C      11=002D      20=002E      60=002F     100=0030      50=0031
     2=0032       3=0033      39=0034      41=0035

CORE REQUIREMENTS FOR SPECL
 COMMON       0  INSKEL COMMON     464  VARIABLES      30  PROGRAM     340


 END OF COMPILATION


 SPECL
 DUP FUNCTION COMPLETED
 // XEQ SPECL L
 *FILES(1,FILE1,1)
 *FILES(2,FILE2,1)
 *CCEND

 CLB, BUILD SPECL

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0051
 *VTV TABLE 3F0C  002A
 *PNT TABLE 3F36  0004
 *DFT TABLE 3F3A  000C
 MAIN SPECL 3F79
 PNT  SPECL 3F38
 LIBF EBPRT 40AC  3F0C
 LIBF SNR   414C  3F0F
 LIBF MDWRT 42B7  3F12
 LIBF MDAF  416B  3F15
 LIBF MDAI  4178  3F18
 LIBF MDCOM 422E  3F1B
 LIBF STFAC 44EE  3F1E
 LIBF SBFAC 44F2  3F21
 LIBF MDI   4170  3F24
 LIBF MWRT  4690  3F27
 LIBF MIOI  4746  3F2A
 LIBF MCOMP 471D  3F2D
 LIBF MIOIX 4752  3F30
 CALL PRT   4B42
 CALL BT2BT 4B8C
 CALL SAVE  4BA8
 CALL IOFIX 4C0C
 LIBF IOU   4C3C  3F33
 CALL BT1BT 4C72
 CORE       4CD8  3328

 CLB, SPECL LD XQ
```

45 Continued

```
  1    0   20
  0    0    0    0    0    0   70   30
  2    0   20
  0    0    0    0    0    0   30   70
  3    0   20
  0    0    0    0    0    0   70   30
  4    0   20
  0    0    0    0    0    0   30   70
  5    0   20
  0    0    0    0    0    0   70   30
  6    0   20
  0    0    0    0    0    0   30   70
```

```
 95    0   20
  0    0    0    0    0    0   70   30
 96    0   20
  0    0    0    0    0    0   30   70
 97    0   20
  0    0    0    0    0    0   70   30
 98    0   20
  0    0    0    0    0    0   30   70
 99    0   20
  0    0    0    0    0    0   70   30
100    0   20
  0    0    0    0    0    0   30   70
```

46

```
// JOB         A
// *  NONPROCESS CORE LOAD
// FOR SCALB
*LIST ALL
**    CALIBRATION PROGRAM FOR SET POINT STATIONS
*IOCS(DISK,1443 PRINTER,CARD)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
C     THIS NONPROCESS CORE LOAD IS FOR CALIBRATING THE SET POINT
C     STATIONS.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      DEFINE FILE 1(2,320,U,II)
  100 READ(2,1)I
    1 FORMAT(I1)
      IF (I)100,100,2
    2 J=9-I
      K=4095+I
      IF(J)100,100,3
    3 WRITE(3,4)I
    4 FORMAT(' SET LOOP'I2,' TO LOW VALUE')
      PAUSE
      CALL AIP(11000,IL,K)
    5 CALL AIP(0,II)
      GO TO (5,6),II
    6 WRITE(3,999)IL
      WRITE(3,7)
    7 FORMAT(' NOW SET IT TO HIGH VALUE')
      PAUSE
      CALL AIP(11000,IH,K)
    8 CALL AIP(0,II)
      GO TO (8,9),II
    9 RANGE(J)=IH-IL
      WRITE(3,999)IH
  999 FORMAT(' VALUE IS'I10)
      LOW(J)=IL
      WRITE(1'1)RANGE,LOW,A,B
      GO TO 100
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF       CSW1(I*)=FFFE       CSW2(I*)=FFFD       CSW3(I*)=FFFC       CSW4(I*)=FFFB       CSW5(I*)=FFFA
   DAY(I*)=FFF9       JOBN(I*)=FFF8      VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8  LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8   AHL(R*)=FFB6-FF68   ALL(R*)=FF66-FF18    A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77      IBASZ(I*)=FE76      IBAZZ(I*)=FE75          G(R*)=FE72-FE54    H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32      ITCNT(I*)=FE31      IPONT(I*)=FE30         II(I )=0006         I(I )=0007         J(I )=0008
     K(I )=0009         IL(I )=000A         IH(I )=000B
```

(46) **Continued**

```
STATEMENT ALLOCATIONS
 1     =0017  4     =0019  7     =0029  999  =0038  100  =0040  2     =004A  3     =005A  5     =0067  6     =0071  8     =0082
 9     =008C

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 AIP      FSTOX    FLOAT    COMGO    ISTOX    MRED     MWRT     MCOMP    MIOI     SUBSC    PAUSE    MDWRT    MDCOM    MDAI     MDAF
 HOLEB    PRNTN    EBPRT    CARDN

INTEGER CONSTANTS
     2=000E       9=000F    4095=0010       3=0011  11000=0012       0=0013       1=0014       0=0015       0=0016

CORE REQUIREMENTS FOR SCALB
 COMMON       0  INSKEL COMMON    464  VARIABLES     14  PROGRAM    172


 END OF COMPILATION



 SCALB
 DUP FUNCTION COMPLETED
 // DUP
 *STORECIL          1 SCALB SCALB
 *FILES(1,FILE1,1)
 *CCEND

 CLB, BUILD SCALB

 CORE LOAD  MAP
 TYPE NAME  ARG1  ARG2

 *CDW TABLE 3E82  000C
 *IBT TABLE 3E8E  001D
 *FIO TABLE 3EAB  0010
 *ETV TABLE 3EBB  0051
 *VTV TABLE 3F0C  0027
 *PNT TABLE 3F34  0004
 *DFT TABLE 3F38  0006
 MAIN SCALB 3F78
 PNT  SCALB 3F36
 LIBF HOLEB 3FF2  3F0C
 LIBF EBPRT 4114  3F0F
 LIBF MRED  432B  3F12
 LIBF MIOI  43F4  3F15
 LIBF MCOMP 43CB  3F18
 LIBF MWRT  433E  3F1B
 LIBF PAUSE 47F0  3F1E
 CALL AIP   4804
 LIBF MDWRT 4983  3F21
 LIBF MDAF  4837  3F24
 LIBF MDAI  4844  3F27
 LIBF MDCOM 48FA  3F2A
 CALL PRT   4BA4
 LIBF IOU   4BEE  3F2D
 CALL IOFIX 4C88
 CALL BT1BT 4CB8
 CALL SAVE  4C24
 LIBF AIPTN 4D1C  3F30
 CALL BT2BT 4D9E
 CORE       4DBC  3244

 CLB, SCALB LD XQ

DUP FUNCTION COMPLETED
```

(47)

```
 // JOB        A
 // *  DATA CARDS FOR CALIBRATING LOOPS 7 AND 8
 // XEQ SCALB   FX

 SET LOOP 7 TO LOW VALUE
 VALUE IS      -202
 NOW SET IT TO HIGH VALUE
 VALUE IS    -31218
 SET LOOP 8 TO LOW VALUE
 VALUE IS      -644
 NOW SET IT TO HIGH VALUE
 VALUE IS    -31522
```

48

```
// JOB         A
// *  NONPROCESS CORE LOAD
// FOR RCALB
*LIST ALL
**    CALIBRATION PROGRAM FOR OP-GUIDE POINTS
*IOCS(DISK,1443 PRINTER,CARD)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
C      THIS NONPROCESS CORE LOAD IS FOR CALIBRATING THE ANALOG INPUT
C      POINTS FOR OP-GUIDE.
C
       INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
       INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
       DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
       COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
      1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
      2,IPONT
       DEFINE FILE 1(2,320,U,II)
   100 READ(2,1)I,ENGH,ENGL
     1 FORMAT(I2,8XF10.2,F10.2)
       IF(I)100,100,2
     2 L=41-I
       M=I-1
       IF(L)100,100,3
     3 WRITE(3,4)I
     4 FORMAT(' SET POINT'I3,' TO LOW VALUE')
       PAUSE
       CALL AIP(11000,IL,M)
     5 CALL AIP(0,II)
       GO TO (5,6),II
     6 WRITE(3,999)IL
       WRITE(3,7)
     7 FORMAT(' NOW SET IT TO HIGH VALUE')
       PAUSE
       CALL AIP(11000,IH,M)
     8 CALL AIP(0,II)
       GO TO (8,9),II
     9 A(L)=(ENGH-ENGL)/(IH-IL)
       B(L)=ENGH-(A(L)*IH)
       WRITE(3,999)IH
   999 FORMAT(' VALUE IS'I10)
       WRITE(1'1)RANGE,LOW,A,B
       GO TO 100
       END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFFB        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18      A(R*)=FF16-FEC8      B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54      H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30        ENGH(R )=0006        ENGL(R )=0008           II(I )=000C
     I(I )=000D           L(I )=000E            M(I )=000F          IL(I )=0010          IH(I )=0011

STATEMENT ALLOCATIONS
 1     =001C  4     =0021  7     =0031  999   =0040  100   =0048  2     =0056  3     =0066  5     =0073  6     =007D  8     =008E
 9     =0098

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 AIP      FSUB     FMPYX    FLD      FSTO     FSTOX    FSBR     FDVR     FLOAT    COMGO    MRED     MWRT     MCOMP    MIOF     MIOI
 SUBSC    PAUSE    MDWRT    MDCOM    MDAI     MDAF     HOLEB    PRNTN    EBPRT    CARDN

INTEGER CONSTANTS
     2=0014      41=0015       1=0016       3=0017  11000=0018       0=0019       0=001A       0=001B

CORE REQUIREMENTS FOR RCALB
 COMMON       0  INSKEL COMMON    464  VARIABLES      20  PROGRAM     188


 END OF COMPILATION


RCALB
DUP FUNCTION COMPLETED
// DUP
*DELET                RCALB
RCALB
D25 NAME NOT IN L/F
*STORECIL           1 RCALB RCALB
*FILES(1,FILE1,1)
*CCEND
```

(48) Continued

```
CLB, BUILD RCALB

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  0030
*PNT TABLE 3F3C  0004
*DFT TABLE 3F40  0006
MAIN RCALB 3F88
PNT  RCALB 3F3E
LIBF HOLEB 4010  3F0C
LIBF EBPRT 4132  3F0F
LIBF MRED  4349  3F12
LIBF MIOI  4412  3F15
LIBF MIOF  440D  3F18
LIBF MCOMP 43E9  3F1B
LIBF MWRT  435C  3F1E
LIBF PAUSE 480E  3F21
CALL AIP   4822
LIBF FSUB  4862  3F24
LIBF FSBR  484E  3F27
LIBF MDWRT 4A3F  3F2A
LIBF MDAF  48F3  3F2D
LIBF MDAI  4900  3F30
LIBF MDCOM 49B6  3F33
CALL PRT   4C60
LIBF IOU   4CAA  3F36
CALL IOFIX 4D44
CALL BT1BT 4D74
CALL SAVE  4CE0
LIBF AIPTN 4DD8  3F39
CALL BT2BT 4E5A
CORE       4E78  3188

CLB, RCALB LD XQ

DUP FUNCTION COMPLETED
```

(49)

```
// JOB         A
// *  DATA CARDS FOR CALIBRATING POINTS 5 THROUGH 12
// XEQ RCALB    FX

SET POINT  5 TO LOW VALUE
VALUE IS       -100
NOW SET IT TO HIGH VALUE
VALUE IS     -32362
SET POINT  6 TO LOW VALUE
VALUE IS       -148
NOW SET IT TO HIGH VALUE
VALUE IS     -32462
SET POINT  7 TO LOW VALUE
VALUE IS       -126
NOW SET IT TO HIGH VALUE
VALUE IS     -32338
SET POINT  8 TO LOW VALUE
VALUE IS       -248
NOW SET IT TO HIGH VALUE
VALUE IS     -32562
SET POINT  9 TO LOW VALUE
VALUE IS       -102

NOW SET IT TO HIGH VALUE

VALUE IS     -32546
SET POINT 10 TO LOW VALUE

VALUE IS        -78
NOW SET IT TO HIGH VALUE
VALUE IS     -32458
SET POINT 11 TO LOW VALUE
VALUE IS        -86
NOW SET IT TO HIGH VALUE
VALUE IS     -32466
SET POINT 12 TO LOW VALUE
VALUE IS       -100
NOW SET IT TO HIGH VALUE
VALUE IS     -32516
```

50
```
// JOB         A
// *  NONPROCESS CORE LOAD
// FOR CMIPT
*LIST ALL
**    CALIBRATION PROGRAM FOR DATA ENTRY DIALS
*IOCS(DISK)
*IOCS(1443 PRINTER)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
C     THIS NONPROCESS CORE LOAD IS FOR CALIBRATING THE DATA ENTRY
C     DIALS.
C
      INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
      INTEGER VALUE(15),SETPT(8),COUMT(8),OFFLN(8)
      DIMENSION INA(16),INB(16)
      DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
      COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
     1PT,COUMT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
     2,IPONT
      DEFINE FILE 1(2,320,U,II)
      WRITE(3,1)
    1 FORMAT(' SET DATA ENTRY DIALS TO 0')
      PAUSE
      CALL GETVL(INA)
      WRITE(3,2)
    2 FORMAT(' SET DATA ENTRY DIALS TO 10')
      PAUSE
      CALL GETVL(INB)
      DO 10 J=1,16
      G(J)=10./(INB(J)-INA(J))
   10 H(J)=10.-(G(J)*INB(J))
      WRITE(1'2)G,H
      CALL EXIT
      END
VARIABLE ALLOCATIONS
  CSW0(I*)=FFFF        CSW1(I*)=FFFE        CSW2(I*)=FFFD        CSW3(I*)=FFFC        CSW4(I*)=FFF8        CSW5(I*)=FFFA
   DAY(I*)=FFF9        JOBN(I*)=FFF8       VALUE(I*)=FFF7-FFE9 RANGE(R*)=FFE6-FFD8   LOW(I*)=FFD7-FFD0 SETPT(I*)=FFCF-FFC8
 COUMT(I*)=FFC7-FFC0 OFFLN(I*)=FFBF-FFB8    AHL(R*)=FFB6-FF68    ALL(R*)=FF66-FF18     A(R*)=FF16-FEC8     B(R*)=FEC6-FE78
 IBASE(I*)=FE77       IBASZ(I*)=FE76       IBAZZ(I*)=FE75          G(R*)=FE72-FE54     H(R*)=FE52-FE34 IENDT(I*)=FE33
 IPERD(I*)=FE32       ITCNT(I*)=FE31       IPONT(I*)=FE30        INA(I )=0017-0008   INB(I )=0027-0018    II(I )=0028
     J(I )=0029

STATEMENT ALLOCATIONS
 1    =0034  2    =0043  10   =007E

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 GETVL   FMPY    FLDX    FSTO    FSTOX   FSBR    FDVR    FLOAT   MWRT    MCOMP   SUBSC   PAUSE   MDWRT   MDCOM   MDAF
 PRNTN   EBPRT

REAL CONSTANTS
  .100000E 02=002C

INTEGER CONSTANTS
     3=002E      1=002F     16=0030      2=0031      0=0032      0=0033

CORE REQUIREMENTS FOR CMIPT
 COMMON       0  INSKEL COMMON    464  VARIABLES     44  PROGRAM    122


 END OF COMPILATION



CMIPT
DUP FUNCTION COMPLETED
*STORECIL         1 CMIPT CMIPT
*FILES(1,FILE1,1)
*CCEND

CLB, BUILD CMIPT

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  001E
*PNT TABLE 3F2A  0004
*DFT TABLE 3F2E  0006
MAIN CMIPT 3F81
```

(50) Continued

```
PNT  CMIPT 3F2C
LIBF EBPRT 3FD4  3F0C
LIBF MWRT  41FE  3F0F
LIBF MCOMP 428B  3F12
LIBF PAUSE 46B0  3F15
CALL GETVL 46EE
LIBF FSBR  478A  3F18
LIBF MDWRT 497B  3F1B
LIBF MDAF  482F  3F1E
LIBF MDCOM 48F2  3F21
CALL PRT   4B9C
LIBF IOU   4BE6  3F24
CALL IOFIX 4C80
CALL BT1BT 4CB0
CALL SAVE  4C1C
LIBF SUBIN 4D14  3F27
CALL BT2BT 4D4E
CORE       4D6C  3294

CLB, CMIPT LD XQ

DUP FUNCTION COMPLETED
```

(51)

```
// JOB         A
// XEQ CMIPT   FX

SET DATA ENTRY DIALS TO 0
SET DATA ENTRY DIALS TO 10
```

(52)

```
// JOB         A
// *  NONPROCESS CORE LOAD
// FOR LOADJ
*LIST ALL
**     PROGRAM TO LOAD JOB DATA FILES ON DISK FROM CARDS
*IOCS(DISK,1443 PRINTER,CARD)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
C      THIS PROGRAM LOADS THE PROCESS JOB FILES ON DISK WITH DATA
C      READ FROM CARDS.
C
       INTEGER SETPT(8)
       DIMENSION AHL(40),ALL(40),INPP(8)
       DEFINE FILE 2(100,320,U,III)
     1 CALL PTIME(TIME)
       WRITE(3,2)TIME
     2 FORMAT('1PROCESS JOB FILE LOAD    TIME'F7.2)
       JOBN=0
       ITIME=0
       DO 3 J=1,8
     3 SETPT(J)=0
       DO 4 J=1,40
       AHL(J)=32000.
     4 ALL(J)=-32000.
    10 READ(2,11)J,IV,BB,CC
    11 FORMAT(I1,4XI5,2F10.0)
       GO TO (100,200,300,400,700),J
   100 IF(IV)600,600,101
   101 JOBN=IV
       ITIME=BB*100.
       I=ITIME/100*100
       K=ITIME-I
       I=I/100
       IF(K-60)102,600,600
   102 AB=I*3600.+K*60.
       IF(32000.-AB)600,103,103
   103 ITIME=AB
       AB=BB
       GO TO 10
   200 IF(IV)600,600,201
   201 I=41-IV
       IF(I)600,600,202
   202 AHL(I)=BB
       ALL(I)=CC
       IF(AHL(I)-ALL(I))600,600,10
   300 IF(IV)600,600,301
```

(52) Continued

```
301 I=9-IV
    IF(I)600,600,302
302 SETPT(I)=BB
    IF(SETPT(I))600,600,303
303 IF(100-SETPT(I))600,600,10
400 IF(JOBN)600,600,409
409 IN=JOBN/100*100
    I=JOBN-IN
    IF(I)600,401,402
401 I=100
402 WRITE(2'I)JOBN,ITIME,SETPT,AHL,ALL
    WRITE(3,403)JOBN,AB
403 FORMAT(' GRADE NUMBER'I6,5X' PRODUCTION TIME'F10.2)
    WRITE(3,500)
500 FORMAT(' OP-GUIDE LIMITS FOR GRADE')
    WRITE(3,501)
501 FORMAT(' POINT   HIGH LIMIT    LOW LIMIT',10X' POINT   HIGH LIMIT
  1   LOW LIMIT')
    DO 503 J=1,39,2
    K=41-J
    J1=J
    J2=J+1
503 WRITE(3,502)J1,AHL(K),ALL(K),J2,AHL(K-1),ALL(K-1)
502 FORMAT(I6,2F13.2,10XI6,2F13.2)
    WRITE(3,510)
510 FORMAT(' CLOSED LOOP SET POINTS FOR GRADE')
    WRITE(3,511)
511 FORMAT(3X'POINT',5X'SETPT',5X'POINT',5X'SETPT',5X'POINT',5X'SETPT'
  1,5X'POINT',5X'SETPT')
    DO 6 J=1,8
  6 INPP(J)=SETPT(J)
    DO 30 I=1,5,4
    J1=I
    J2=I+1
    J3=I+2
    J4=I+3
    IA=9-I
    IB=8-I
    IC=7-I
    ID=6-I
 30 WRITE(3,7)J1,INPP(IA),J2,INPP(IB),J3,INPP(IC),J4,INPP(ID)
  7 FORMAT(I8,I10,3(I10,I10))
    GO TO 1
600 WRITE(3,601)J,IV,BB,CC
601 FORMAT(' INVALID DATA CARD',/I5,4XI5,2F10.2)
    GO TO 1
700 CALL EXIT
    END
VARIABLE ALLOCATIONS
   AHL(R )=0054-0006   ALL(R )=00A4-0056 CTIME(R )=00A6         BB(R )=00A8         CC(R )=00AA          AB(R )=00AC
  INPP(I )=00B7-00B0 SETPT(I )=00BF-00B8    III(I )=00C0       JOBN(I )=00C1      ITIME(I )=00C2  •        J(I )=00C3
    IV(I )=00C4           I(I )=00C5           K(I )=00C6         IN(I )=00C7         J1(I )=00C8          J2(I )=00C9
    J3(I )=00CA          J4(I )=00CB          IA(I )=00CC         IB(I )=00CD         IC(I )=00CE          ID(I )=00CF

STATEMENT ALLOCATIONS
 2     =00EB  11    =00FD  403   =0103  500   =0118  501   =0127  502   =014B  510   =0153  511   =0166  7     =018F  601   =0196
 1     =01A7  3     =01BC  4     =01DB  10    =01EB  100   =0200  101   =0204  102   =022C  103   =0243  200   =024E  201   =0252
 202   =025C  300   =0276  301   =027A  302   =0284  303   =0294  400   =02A1  409   =02A5  401   =02BB  402   =02BF  503   =02F4
 6     =0324  30    =0368  600   =039C  700   =03AA

FEATURES SUPPORTED
 NONPROCESS
 ONE WORD INTEGERS
 IOCS

CALLED SUBPROGRAMS
 PTIME    FADD     FSUB     FSUBX    FMPY     FLD      FLDX     FSTO     FSTOX    IFIX     FLOAT    COMGO    ISTOX    LDFAC    MRED
 MWRT     MCOMP    MIOFX    MIOIX    MIOF     MIOI     SUBSC    SNR      MDWRT    MDCOM    MDAI     MDAF     MDI      HOLEB    PRNTN
 EBPRT    CARDN

REAL CONSTANTS
  .320000E 05=00D4    .100000E 03=00D6    .360000E 04=00D8    .600000E 02=00DA

INTEGER CONSTANTS
     3=00DC       0=00DD       1=00DE       8=00DF      40=00E0       2=00E1     100=00E2      60=00E3      41=00E4        9=00E5
    39=00E6       5=00E7       4=00E8       7=00E9       6=00EA

CORE REQUIREMENTS FOR LOADJ
 COMMON       0  INSKEL COMMON       0  VARIABLES    212  PROGRAM    728


 END OF COMPILATION
```

52 Continued

```
LOADJ
DUP FUNCTION COMPLETED
*DELET              LOADJ
LOADJ
D25 NAME NOT IN L/F
*STORECIL          1 LOADJ LOADJ
*FILES(2,FILE2,1)
*CCEND

CLB, BUILD LOADJ

CORE LOAD  MAP
TYPE NAME  ARG1  ARG2

*CDW TABLE 3E82  000C
*IBT TABLE 3E8E  001D
*FIO TABLE 3EAB  0010
*ETV TABLE 3EBB  0051
*VTV TABLE 3F0C  003F
*PNT TABLE 3F4C  0004
*DFT TABLE 3F50  0006
MAIN LOADJ 40F7
PNT  LOADJ 3F4E
LIBF HOLEB 42FC  3F0C
LIBF EBPRT 441E  3F0F
CALL PTIME 44CA
LIBF MWRT  4684  3F12
LIBF MIOF  4735  3F15
LIBF MCOMP 4711  3F18
LIBF SNR   4B36  3F1B
LIBF MRED  4671  3F1E
LIBF MIOI  473A  3F21
LIBF FADD  4B6E  3F24
LIBF FSUB  4B62  3F27
LIBF LDFAC 4BEC  3F2A
LIBF FSUBX 4B5D  3F2D
LIBF MDWRT 4D6D  3F30
LIBF MDI   4C26  3F33
LIBF MDAI  4C2E  3F36
LIBF MDAF  4C21  3F39
LIBF MDCOM 4CE4  3F3C
LIBF MIOFX 4741  3F3F
LIBF MIOIX 4746  3F42
CALL PRT   4F8E
LIBF SUBIN 4FD8  3F45
LIBF IOU   5012  3F48
CALL IOFIX 50AC
CALL BT1BT 50DC
CALL SAVE  5048
CALL BT2BT 5140
CORE       515E  2EA2

CLB, LOADJ LD XQ

DUP FUNCTION COMPLETED
```

53

```
// JOB         A
// *  UPDATE A PROCESS JOB FILE ON DISK
// XEQ LOADJ    FX
PROCESS JOB FILE LOAD    TIME  14.58
GRADE NUMBER 12345       PRODUCTION TIME       1.30
OP-GUIDE LIMITS FOR GRADE
POINT   HIGH LIMIT    LOW LIMIT             POINT   HIGH LIMIT    LOW LIMIT
    1     32000.00    -32000.00                 2     32000.00    -32000.00
    3     32000.00    -32000.00                 4     32000.00    -32000.00
    5         5.45         4.96                 6         5.45         4.96
    7         5.45         4.96                 8         5.45         4.96
    9         5.45         4.96                10         5.45         4.96
   11         5.45         4.96                12         5.45         4.96
   13     32000.00    -32000.00                14     32000.00    -32000.00
   15     32000.00    -32000.00                16     32000.00    -32000.00
   17     32000.00    -32000.00                18     32000.00    -32000.00
   19     32000.00    -32000.00                20     32000.00    -32000.00
   21     32000.00    -32000.00                22     32000.00    -32000.00
   23     32000.00    -32000.00                24     32000.00    -32000.00
   25     32000.00    -32000.00                26     32000.00    -32000.00
   27     32000.00    -32000.00                28     32000.00    -32000.00
   29     32000.00    -32000.00                30     32000.00    -32000.00
   31     32000.00    -32000.00                32     32000.00    -32000.00
   33     32000.00    -32000.00                34     32000.00    -32000.00
   35     32000.00    -32000.00                36     32000.00    -32000.00
```

53 **Continued**

```
   37      32000.00     -32000.00                38      32000.00     -32000.00
   39      32000.00     -32000.00                40      32000.00     -32000.00
CLOSED LOOP SET POINTS FOR GRADE
  POINT      SETPT      POINT      SETPT      POINT      SETPT      POINT      SETPT
      1          0          2          0          3          0          4          0
      5          0          6          0          7         15          8         86
PROCESS JOB FILE LOAD    TIME  14.58
```

54
```
// JOB          A
// DUP
*DUMPLET
```

```
      LET

PACK LABEL
 00000

.DCOM  0010   0000     .MBT   0020   0010     .SKSB  0020   0030     .SUP   0080   0050     .CLB   00A0   0100     .DUP   0440   01A0
.ASM   0300   05E0     .FOR   0680   08E0     .SIM   05F0   0F60     .LET   0080   1550     IAND   0002   15D0     CLEAR  0009   15D2
CLOCK  0002   15DB     COUNT  0004   15DD     DMP    0017   15E1     DMPHX                   DMPDC                   DMPS   0010   15F8
DMPST                  DPART  0002   1608     ENDTS  0002   160A     IEOR   0002   160C     LD     0002   160E     LEVEL  0004   1610
MASK   0003   1614     OPMON  0002   1617     IOR    0002   1619     QIFON  000A   161B     QUEUE  000C   1625     RESMK  0004   1631
SAVMK  0003   1635     SETCL  0003   1638     TIMER  0006   163B     UNMK   0005   1641     UNQ    0005   1646     VIAQ   0007   164B
CONHX  0006   1652     TRPRT  0007   1658     FLIP   0007   165F     EADD   000B   1666     ESUB                    EADDX
ESUBX                  ESBR                   ESBRX                  EATN   000D   1671     EATAN                   EAVL   0003   167E
EABS                   EAXB   0006   1681     EAXBX                  EAXI   0006   1687     EAXIX                   EDVR   0007   168D
EDVRX                  EDIV                   EDIVX                  ELD    0009   1694     ELDX                    ESTO
ESTOX                  ELN    000B   169D     EALOG                  EMPY   0004   16A8     EMPYX                   ESINE  000D   16AC
ESIN                   ECOSN                  ECOS                   ESQR   0007   16B9     ESQRT                   ETNH   0006   16C0
ETANH                  ETRTN  0004   16C6     ETNTR                  EXPN   000B   16CA     EEXP                    FSBR   000B   16D5
FSBRX                  FADD                   FSUB                   FADDX                   FSUBX                   FARC   0004   16E0
FATN   000C   16E4     FATAN                  FAVL   0003   16F0     FABS                    FAXB   0006   16F3     FAXBX
FAXI   0006   16F9     FAXIX                  FBTD   001A   16FF     FDTB                    FDIV   0008   1719     FDIVX
FDVR                   FDVRX                  FIXIX  0005   1721     FIXI                    FLD    0009   1726     FLDX
FSTO                   FSTOX                  FLN    000B   172F     FALOG                   FLOAT  0003   173A     FMPY   0005   173D
FMPYX                  FSINE  000B   1742     FSIN                   FCOSN                   FCOS                    FSQR   0007   174D
FSQRT                  FTNH   0006   1754     FTANH                  FTRTN  0004   175A     FTNTR                   FXPN   0009   175E
FEXP                   IABS   0003   1767     IFIX   0004   176A     NORM   0004   176E     SNR    0003   1772     XDD    0006   1775
XMD    0005   177B     XMDS   0004   1780     XSQR   0004   1784     BINDC  0006   1788     BINHX  0004   178E     DCBIN  0006   1792
EBPA   0006   1798     EBPRT  000A   179E     HOLEB  0012   17A8     HOLPR  000D   17BA     HXBIN  0005   17C7     PAPEB  0010   17CC
PAPHL  0014   17DC     PAPPR  0011   17F0     PRT    0005   1801     ADRCK  0007   1806     COMGO  0006   180D     COMG1
DATSW  0004   1813     DVCHK  0002   1817     ESIGN  0005   1819     FCTST  0003   181E     FSIGN  0005   1821     IOU    0007   1826
ISIGN  0003   182D     ISTOX  0003   1830     LDFAC  0004   1833     STFAC                   SBFAC                   DVFAC
MDFIO  0023   1837     MDAF                   MDAI                   MDCOM                   MDF                     MDFX
MDI                    MDIX                   MDRED                  MDWRT                   MDFND  0008   185A     MFIO   0059   1862
MRED                   MWRT                   MCOMP                  MIOAF                   MIOAI                   MIOFX
MIOIX                  MIOF                   MIOI                   MGOTO  000E   18BB     MFIF                    MIIF
MEIF                   MIAR   000E   18C9     MIARX                  MFAR                    MFARX                   MEAR
MEARX                  OVERF  0002   18D7     PAUSE  0002   18D9     REWND  0009   18DB     BCKSP                   EOF
SAVE   000A   18E4     IOFIX                  SLITE  0006   18EE     SLITT                   SSWTC  0004   18F4     STOP   0003   18F8
SUBIN  0005   18FB     SUBSC  0004   1900     TSTOP  0002   1904     TSTRT  0002   1906     TTEST  0003   1908     TSET
UFIO   001C   190B     URED                   UWRT                   UIOI                    UIOF                    UIOAI
UIOAF                  UIOFX                  UIOIX                  UCOMP                   PLOTX  000D   1927     CARDN  0016   1934
PAPTN  0010   194A     MAGT   0020   195A     AIPTN  0009   197A     AIPN                    AISQN  000F   1983     AISN
AIRN   000D   1992     ANINT  0014   199F     DINP   0013   19B3     DIEXP  0006   19C6     DICMP  0007   19CC     DAOP   0013   19D3
IOPE   0009   19E6     OUSLY                  ETS                    XSAVE  0009   19EF     XEXIT                   XLOAD
GAGED  0003   19F8     UNGAG                  AIP    0004   19FB     AIS    000D   19FF     AIR    0011   1A0C     CS     0008   1A1D
VS                     DI                     PI                     CSC    000A   1A25     VSC                     DIC
PIC                    CSX    0004   1A2F     VSX                    DIX                     PIX                     DAC    0007   1A33
CO                     DO                     PO                     QZERQ  0002   1A3A     QZ010  0006   1A3C     BT1BT  0007   1A42
BT2BT  0003   1A49     FCHAR  0005   1A4C     SCALF  0002   1A51     FGRID  0007   1A53     FPLOT  0004   1A5A     ECHAR  0005   1A5E
SCALE  0002   1A63     EGRID  0008   1A65     EPLOT  0005   1A6D     POINT  0007   1A72     FCHRX  0024   1A79     FCHRI
WCHRI                  FRULE  0009   1A9D     FMOVE                  FINC                    ECHRX  0025   1AA6     ECHRI
VCHRI                  ERULE  000B   1ACB     EMOVE                  EINC                    XYPLT  0007   1AD6     PLOTI  0003   1ADD
PLOTS                  .TEMP  1AE0   1B00     .E     5A00   1B00


       FLET

PACK LABEL
 00000

9DUMY  00A0   05A0     .E     00A0   05A0
```

(54) Continued

```
        LET

PACK LABEL
 11111

.LET  0080   0000     SYDIR 009E   0080     OUTTR                   CHAIN                   INTEX                   SHARE
SPECL                 BACK                  EACLK                   SCHED 0014   011E       LEV10 0024   0132       SOUT  0003   0156
QUE15 0002   0159     TCONT 0003   015B     TABRT 0002   015E       GETVL 000B   0160       CONVR 0005   016B       PTIME 0005   0170
IADDR 0002   0175     ISBAD 0002   0177     CESET 0002   0179       ABORT 0002   017B       ENDGD 0002   017D       .TEMP 017F   0180
.E    1180   0180




        FLET

PACK LABEL
 11111

.PRWS 0051   1118     .FIOS 000F   1169     .MESS 00A3   1178       /EPDM 7FFF   121B       /EPSV 0780   1282       /INSV 48FF   1288
/NPSV 4180   12C3     FILE1 0002   12F8     FILE2 0064   12FA       FILE3 0003   135E       COLDN 05DC   1361       COLDS 00D6   1366
RSTAR 0F8C   1367     COLDP 10CE   1374     CEINT 0A26   1382       DUM   012C   138B       IDUM  0094   138C       CDUM  013A   938D
GRADE 116A   138E     SCAN2 0C2E   939C     LOG15 0D5E   13A6       LOG60 0AE0   13B1       SHIFT 0AE0   13BA       WEEK  0ADC   13C3
TREND 0B7A   13CC     COGLP 101E   13D6     CCLSP 0F76   13E3       MGRTP 0FB4   13F0       CPJSP 0FCC   13FD       STRND 0BFC   140A
AIMON 0D90   1414     SCALB 0F3A   141F     RCALB 0FF6   142C       CMIPT 0EEA   1439       LOADJ 12DC   1445       9DUMY 0123   1455
/SPSV 4180   1578     /PRSV 4180   15AD     .SKEL 0036   15E2       .EPRG 0022   1618       /CLST 0780   163A       .E    0280   12F8

DUP FUNCTION COMPLETED

// JOB         A
// END
```

PROGRAM LISTING NO. 10: ON-LINE PROCESS OUTPUT

```
  TURN OFF WRITE STORAGE PROTECT SWITCH
1 ENTER TIME THROUGH DATA SWITCHES
  TIME ENTERED WAS   08.016   HOURS
   PROCESS COLD START
   PRODUCTION STOP    TIME   8.00    DAY 6


      0    0    0    1    2    3    4    5
      0    0    0    0    0    0    0    0
   NEXT JOB 12345  QUEUE SEQUENCE  45  DAY 6  TIME   8.02


   START OF GRADE 12345 PRODUCTION TIME     1.29 START TIME     8.03 DAY  6
   OP-GUIDE LIMITS FOR NEW GRADE
   POINT   HIGH LIMIT    LOW LIMIT              POINT   HIGH LIMIT    LOW LIMIT
       1     32000.00    -32000.00                  2     32000.00    -32000.00
       3     32000.00    -32000.00                  4     32000.00    -32000.00
       5         5.45         4.96                  6         5.45         4.96
       7         5.45         4.96                  8         5.45         4.96
       9         5.45         4.96                 10         5.45         4.96
      11         5.45         4.96                 12         5.45         4.96
      13     32000.00    -32000.00                 14     32000.00    -32000.00
      15     32000.00    -32000.00                 16     32000.00    -32000.00
      17     32000.00    -32000.00                 18     32000.00    -32000.00
      19     32000.00    -32000.00                 20     32000.00    -32000.00
      21     32000.00    -32000.00                 22     32000.00    -32000.00
      23     32000.00    -32000.00                 24     32000.00    -32000.00
      25     32000.00    -32000.00                 26     32000.00    -32000.00
      27     32000.00    -32000.00                 28     32000.00    -32000.00
      29     32000.00    -32000.00                 30     32000.00    -32000.00
      31     32000.00    -32000.00                 32     32000.00    -32000.00
      33     32000.00    -32000.00                 34     32000.00    -32000.00
      35     32000.00    -32000.00                 36     32000.00    -32000.00
      37     32000.00    -32000.00                 38     32000.00    -32000.00
      39     32000.00    -32000.00                 40     32000.00    -32000.00
   CLOSED LOOP SET POINTS FOR NEW GRADE
     POINT      SETPT      POINT      SETPT      POINT      SETPT      POINT      SETPT
         1          0          2          0          3          0          4          0
         5          0          6          0          7         14          8         86

   NORMAL SCAN    DAY  6     TIME         8.03


      7    0    0    0    0    0    5    0
      0    0    0    0    0    0    0    0
   LOOP  7 NEW SET POINT  50 DAY 6 TIME   8.05

   NORMAL SCAN    DAY  6     TIME         8.06

   NORMAL SCAN    DAY  6     TIME         8.07

   NORMAL SCAN    DAY  6     TIME         8.09

   NORMAL SCAN    DAY  6     TIME         8.12

   NORMAL SCAN    DAY  6     TIME         8.13

   LOG15   DAY  6     TIME       8.15
   OP-GUIDE POINTS
     POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
         1   14154.00          2    9734.00          3      24.00          4      32.00
         5       5.37          6       5.16          7       5.01          8       5.10
         9       5.14         10       5.28         11       5.00         12       4.98
        13      22.00         14      20.00         15      22.00         16      42.00
        17     714.00         18     704.00         19     716.00         20     706.00
        21     706.00         22     698.00         23     718.00         24     706.00
        25     712.00         26     708.00         27     704.00         28     714.00
        29     704.00         30     718.00         31     706.00         32     704.00
        33     714.00         34     710.00         35     704.00         36     718.00
        37     710.00         38     704.00         39     716.00         40     706.00
   CLOSED LOOP POINTS
     POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
         1          0          2          0          3         75          4          0
         5          0          6          5          7         49          8         85


   SHIFT END LOG   DAY  6     TIME       8.15
```

① (Continued)

```
NORMAL SCAN   DAY  6    TIME         8.15

NORMAL SCAN   DAY  6    TIME         8.18

NORMAL SCAN   DAY  6    TIME         8.19

NORMAL SCAN   DAY  6    TIME         8.21

NORMAL SCAN   DAY  6    TIME         8.24

NORMAL SCAN   DAY  6    TIME         8.25

NORMAL SCAN   DAY  6    TIME         8.27

NORMAL SCAN   DAY  6    TIME         8.30

LOG15   DAY  6    TIME     8.30
OP-GUIDE POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
      1   14162.00          2    9760.00          3      30.00          4      40.00
      5       5.37          6       5.16          7       5.00          8       5.09
      9       5.14         10       5.27         11       5.00         12       4.99
     13      18.00         14      28.00         15      18.00         16      22.00
     17     710.00         18     708.00         19     706.00         20     710.00
     21     706.00         22     704.00         23     714.00         24     706.00
     25     700.00         26     714.00         27     706.00         28     706.00
     29     702.00         30     712.00         31     706.00         32     710.00
     33     704.00         34     716.00         35     708.00         36     712.00
     37     704.00         38     714.00         39     704.00         40     714.00
CLOSED LOOP POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
      1          0          2          0          3         75          4          0
      5          0          6          5          7         49          8         85

NORMAL SCAN   DAY  6    TIME         8.31

NORMAL SCAN   DAY  6    TIME         8.33

NORMAL SCAN   DAY  6    TIME         8.36

NORMAL SCAN   DAY  6    TIME         8.37

NORMAL SCAN   DAY  6    TIME         8.39

NORMAL SCAN   DAY  6    TIME         8.42

NORMAL SCAN   DAY  6    TIME         8.43

LOG15   DAY  6    TIME     8.45
OP-GUIDE POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
      1   14158.00          2    9770.00          3      28.00          4      34.00
      5       5.37          6       5.16          7       5.01          8       5.10
      9       5.14         10       5.27         11       5.00         12       4.98
     13      24.00         14      20.00         15      20.00         16      18.00
     17     706.00         18     706.00         19     704.00         20     716.00
     21     704.00         22     716.00         23     708.00         24     706.00
     25     706.00         26     708.00         27     704.00         28     716.00
     29     706.00         30     704.00         31     714.00         32     704.00
     33     714.00         34     704.00         35     714.00         36     704.00
     37     714.00         38     704.00         39     718.00         40     706.00
CLOSED LOOP POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
      1          0          2          0          3         75          4          0
      5          0          6          5          7         49          8         85

NORMAL SCAN   DAY  6    TIME         8.45

NORMAL SCAN   DAY  6    TIME         8.48
```

(1) (Continued)

```
NORMAL SCAN   DAY  6    TIME        8.49


NORMAL SCAN   DAY  6    TIME        8.51


NORMAL SCAN   DAY  6    TIME        8.54


NORMAL SCAN   DAY  6    TIME        8.55


NORMAL SCAN   DAY  6    TIME        8.57


NORMAL SCAN   DAY  6    TIME        9.00


LOG15    DAY  6    TIME      9.00
OP-GUIDE POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
      1  14158.00          2   9818.00          3     26.00          4     50.00
      5      5.37          6      5.16          7      5.01          8      5.10
      9      5.14         10      5.27         11      5.00         12      4.98
     13     38.00         14     28.00         15     30.00         16     30.00
     17    704.00         18    718.00         19    706.00         20    708.00
     21    702.00         22    716.00         23    712.00         24    700.00
     25    718.00         26    704.00         27    716.00         28    706.00
     29    710.00         30    710.00         31    706.00         32    706.00
     33    704.00         34    718.00         35    704.00         36    718.00
     37    708.00         38    708.00         39    712.00         40    706.00
CLOSED LOOP POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
      1         0          2         0          3        75          4         0
      5         0          6         5          7        50          8        85


ONE HOUR LOG     DAY   6      TIME       9.00


NORMAL SCAN   DAY  6    TIME        9.01


NORMAL SCAN   DAY  6    TIME        9.03


NORMAL SCAN   DAY  6    TIME        9.06


NORMAL SCAN   DAY  6    TIME        9.07


NORMAL SCAN   DAY  6    TIME        9.09


NORMAL SCAN   DAY  6    TIME        9.12


NORMAL SCAN   DAY  6    TIME        9.13


LOG15    DAY  6    TIME      9.15
OP-GUIDE POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
      1  14150.00          2   9858.00          3     28.00          4     20.00
      5      5.37          6      5.16          7      5.01          8      5.10
      9      5.14         10      5.27         11      5.00         12      4.98
     13     22.00         14     26.00         15     24.00         16     26.00
     17    708.00         18    708.00         19    706.00         20    704.00
     21    714.00         22    706.00         23    708.00         24    706.00
     25    708.00         26    708.00         27    706.00         28    708.00
     29    704.00         30    716.00         31    708.00         32    710.00
     33    706.00         34    706.00         35    704.00         36    714.00
     37    706.00         38    704.00         39    716.00         40    704.00
CLOSED LOOP POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
      1         0          2         0          3        75          4         0
      5         0          6         5          7        49          8        85
```

2

```
   0    0    0    0    0    5    1    0
   5    0    0    0    0    4    9    0
OP-GUIDE PT  5 HIGH LIMIT      5.10 LOW LIMIT      4.90 DAY 6 TIME  11.28


NORMAL SCAN   DAY  6    TIME        11.30
HIGH LIMIT VIOLATION  POINT  5    VALUE      5.3716


LOG15   DAY  6    TIME    11.30
OP-GUIDE POINTS
  POINT     VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
      1   14154.00         2    9260.00          3      26.00          4     106.00
      5       5.37         6       5.15          7       5.01          8       5.09
      9       5.14        10       5.27         11       5.00         12       4.98
     13      26.00        14      38.00         15      24.00         16      20.00
     17     708.00        18     704.00         19     718.00         20     704.00
     21     712.00        22     706.00         23     704.00         24     714.00
     25     706.00        26     706.00         27     704.00         28     714.00
     29     706.00        30     704.00         31     714.00         32     706.00
     33     702.00        34     716.00         35     706.00         36     702.00
     37     714.00        38     704.00         39     712.00         40     710.00
CLOSED LOOP POINTS
  POINT     VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
      1         0          2          0          3          1          4          0
      5         0          6          5          7         65          8         83


DEMAND SCAN   DAY  6    TIME        11.30


NORMAL SCAN   DAY  6    TIME        11.32



   0    0    0    0    0    0    0    5
   0    0    0    0    0    0    0    0
DAY 6 TIME  11.34
OP-GUIDE POINT   VALUE   HIGH LIMIT   LOW LIMIT     FACTOR A         FACTOR B
       5          5.03       5.10        4.90    -0.309962E-03   -0.309947E-01


NORMAL SCAN   DAY  6    TIME        11.34


NORMAL SCAN   DAY  6    TIME        11.36


NORMAL SCAN   DAY  6    TIME        11.38
```

(3)

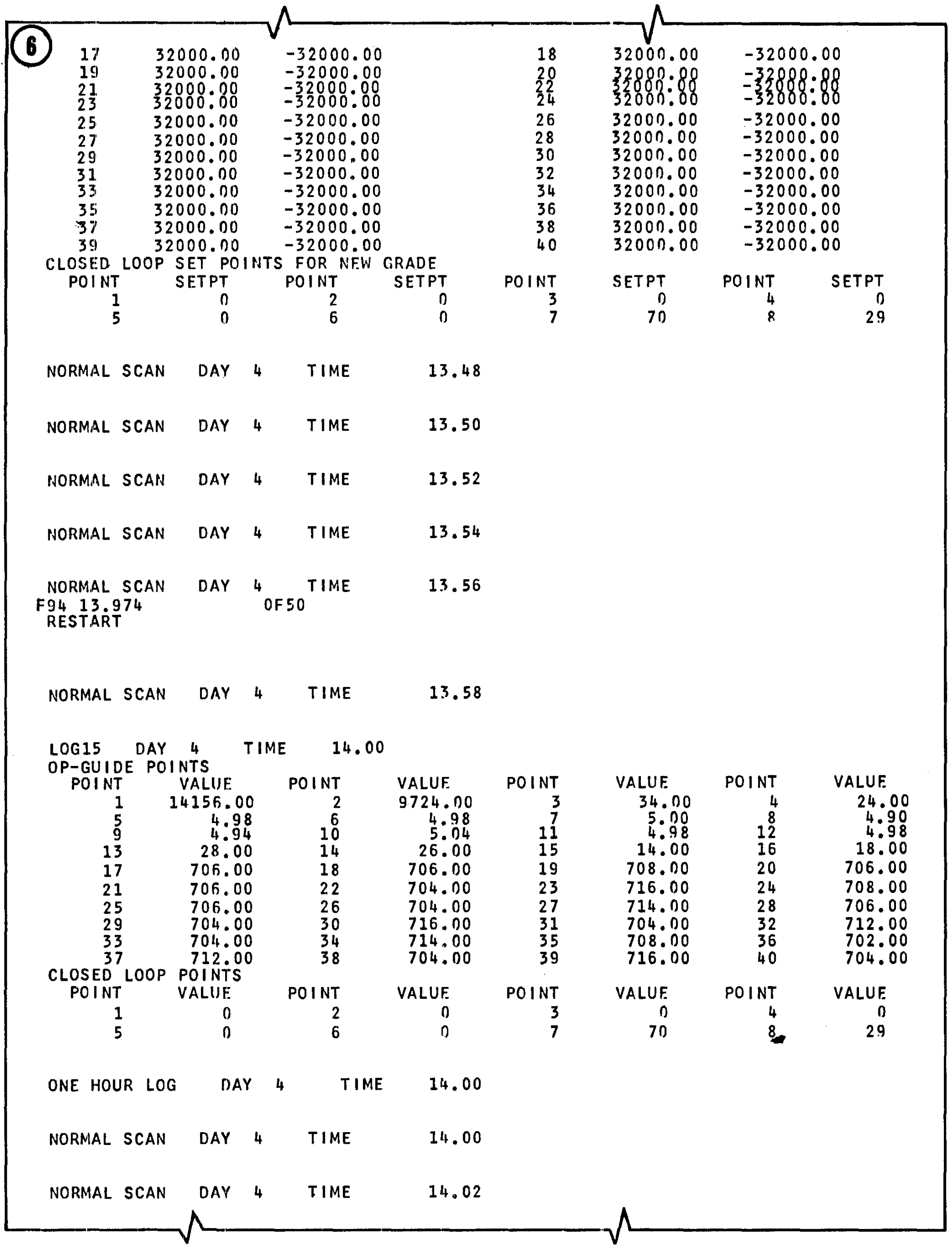
```
   0    0    6    0    0    0    1    0
   0    0    0    0    0    3    0    0
TREND LOG OP-GUIDE POINT  6  PERIOD  10    COUNT  300
TREND OP-GUIDE  6    VALUE      5.01
TREND OP-GUIDE  6    VALUE      5.01


NORMAL SCAN    DAY  3    TIME        9.17
TREND OP-GUIDE  6    VALUE      5.49
TREND OP-GUIDE  6    VALUE      5.65
TREND OP-GUIDE  6    VALUE      5.64
TREND OP-GUIDE  6    VALUE      5.79
TREND OP-GUIDE  6    VALUE      6.19
TREND OP-GUIDE  6    VALUE      6.56
TREND OP-GUIDE  6    VALUE      6.73
TREND OP-GUIDE  6    VALUE      6.73


DEMAND SCAN    DAY  3    TIME        9.19
HIGH LIMIT VIOLATION  POINT  6   VALUE       6.7345
TREND OP-GUIDE  6    VALUE      6.73
TREND OP-GUIDE  6    VALUE      6.42
TREND OP-GUIDE  6    VALUE      5.96


NORMAL SCAN    DAY  3    TIME        9.19
HIGH LIMIT VIOLATION  POINT  6   VALUE       5.5796
TREND OP-GUIDE  6    VALUE      5.57
TREND OP-GUIDE  6    VALUE      5.58
TREND OP-GUIDE  6    VALUE      5.12
TREND OP-GUIDE  6    VALUE      4.98
TREND OP-GUIDE  6    VALUE      4.98



   0    0    0    0    0    0    0    0
   0    0    0    0    0    3    0    0
NEXT JOB     0  QUEUE SEQUENCE   0  DAY 3  TIME   9.21


NORMAL SCAN    DAY  3    TIME        9.21


NORMAL SCAN    DAY  3    TIME        9.23
PRODUCTION STOP    TIME   9.25    DAY 3



   0    0    0    0    0    0    3    5
   0    0    0    0    0    3    0    0
NEXT JOB    35  QUEUE SEQUENCE  35  DAY 3  TIME   9.25
```

(4)

```
   0    0    0    0    4    5    3    0
   0    0    0    0    0    0    0    0
NEXT JOB  4530   QUEUE SEQUENCE  30  DAY 2  TIME  12.19


NORMAL SCAN    DAY  2    TIME        12.20


NORMAL SCAN    DAY  2    TIME        12.22


NORMAL SCAN    DAY  2    TIME        12.24


START OF GRADE  4530 PRODUCTION TIME     1.25 START TIME     12.26 DAY  2
OP-GUIDE LIMITS FOR NEW GRADE
POINT   HIGH LIMIT    LOW LIMIT              POINT   HIGH LIMIT     LOW LIMIT
    1     32000.00    -32000.00                 2     32000.00    -32000.00
    3     32000.00    -32000.00                 4     32000.00    -32000.00
    5         5.55         4.45                 6         5.45         4.45
    7         5.55         4.45                 8         5.45         4.45
    9         5.35         4.45                10         5.65         4.45
   11         5.63         4.45                12         5.62         4.45
   13     32000.00    -32000.00                14     32000.00    -32000.00
   15     32000.00    -32000.00                16     32000.00    -32000.00
   17     32000.00    -32000.00                18     32000.00    -32000.00
```

5

```
   39     32000.00    -32000.00              40     32000.00    -32000.00
 CLOSED LOOP SET POINTS FOR NEW GRADE
   POINT     SETPT     POINT     SETPT     POINT     SETPT     POINT     SETPT
       1         0         2         0         3         0         4         0
       5         0         6         0         7        70         8        29


 NORMAL SCAN   DAY  3    TIME         9.06
 997 09.130         RELOAD
 RELOAD

 PROCESS RESTART CHECK POINT


 NORMAL SCAN   DAY  3    TIME         9.07


 LOG15   DAY  3    TIME      9.07
 OP-GUIDE POINTS
   POINT     VALUE     POINT     VALUE     POINT     VALUE     POINT     VALUE
       1  14150.00         2   9946.00         3     24.00         4   -100.00
       5      5.03         6      5.01         7      4.99         8      4.90
       9      4.94        10      5.04        11      4.98        12      4.98
      13     30.00        14     22.00        15     24.00        16     26.00
      17    702.00        18    714.00        19    706.00        20    704.00
      21    714.00        22    704.00        23    716.00        24    708.00
      25    708.00        26    708.00        27    708.00        28    704.00
      29    716.00        30    704.00        31    712.00        32    702.00
      33    714.00        34    710.00        35    704.00        36    716.00
      37    704.00        38    716.00        39    704.00        40    718.00
 CLOSED LOOP POINTS
   POINT     VALUE     POINT     VALUE     POINT     VALUE     POINT     VALUE
       1         0         2         0         3        75         4         0
       5         0         6         0         7        69         8        29


 ONE HOUR LOG     DAY  3     TIME      9.07


 SHIFT END LOG    DAY  3     TIME      9.08


 NORMAL SCAN   DAY  3    TIME         9.09


 DEMAND SCAN   DAY  3    TIME         9.10
 HIGH LIMIT VIOLATION  POINT  5     VALUE       7.3461


 DEMAND SCAN   DAY  3    TIME         9.10
```

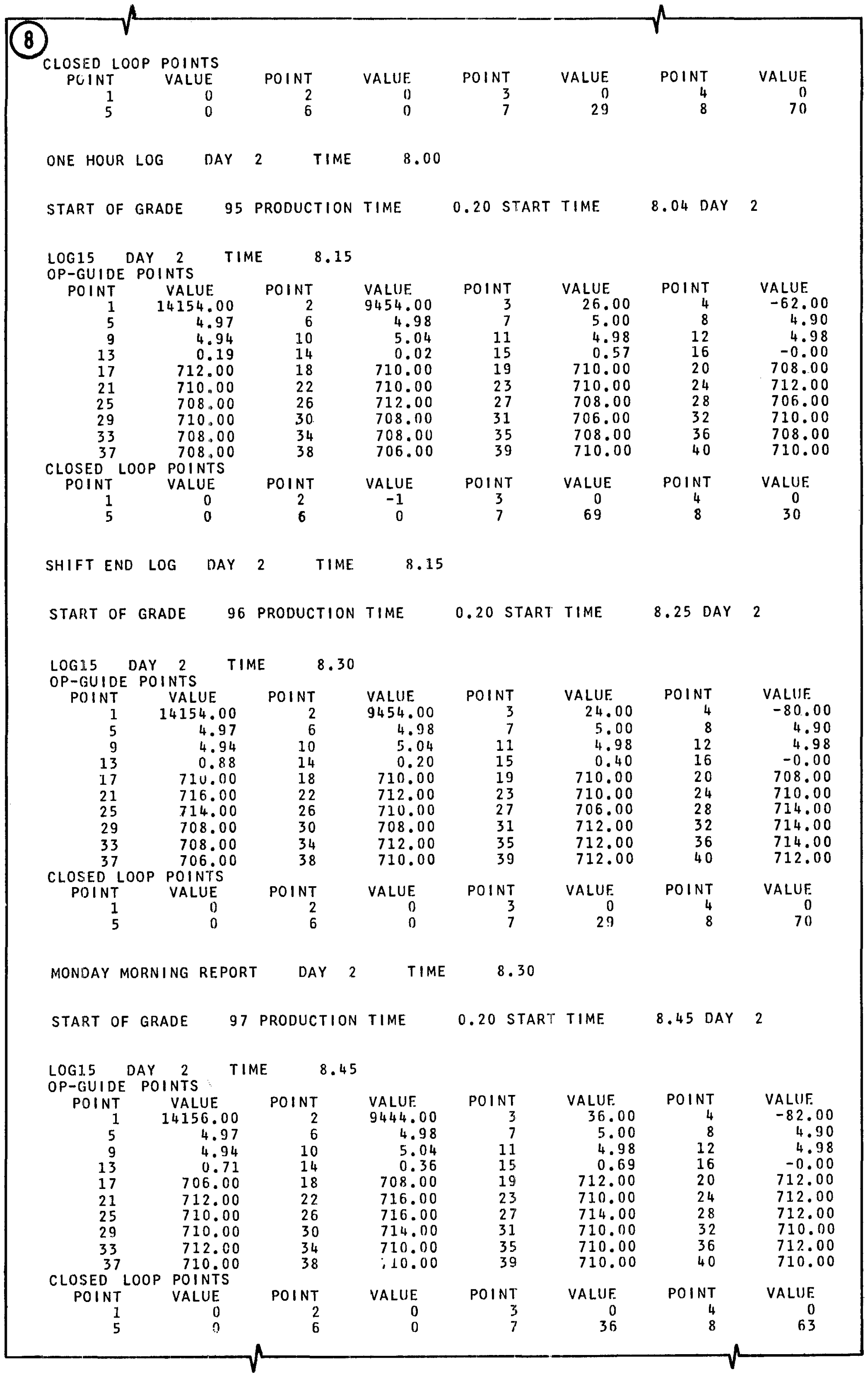


6

```
    17     32000.00     -32000.00                 18     32000.00     -32000.00
    19     32000.00     -32000.00                 20     32000.00     -32000.00
    21     32000.00     -32000.00                 22     32000.00     -32000.00
    23     32000.00     -32000.00                 24     32000.00     -32000.00
    25     32000.00     -32000.00                 26     32000.00     -32000.00
    27     32000.00     -32000.00                 28     32000.00     -32000.00
    29     32000.00     -32000.00                 30     32000.00     -32000.00
    31     32000.00     -32000.00                 32     32000.00     -32000.00
    33     32000.00     -32000.00                 34     32000.00     -32000.00
    35     32000.00     -32000.00                 36     32000.00     -32000.00
    37     32000.00     -32000.00                 38     32000.00     -32000.00
    39     32000.00     -32000.00                 40     32000.00     -32000.00
 CLOSED LOOP SET POINTS FOR NEW GRADE
   POINT      SETPT      POINT      SETPT      POINT      SETPT      POINT      SETPT
       1          0          2          0          3          0          4          0
       5          0          6          0          7         70          8         29


 NORMAL SCAN    DAY  4    TIME        13.48


 NORMAL SCAN    DAY  4    TIME        13.50


 NORMAL SCAN    DAY  4    TIME        13.52


 NORMAL SCAN    DAY  4    TIME        13.54


 NORMAL SCAN    DAY  4    TIME        13.56
F94 13.974          OF50
 RESTART


 NORMAL SCAN    DAY  4    TIME        13.58


 LOG15   DAY  4    TIME    14.00
 OP-GUIDE POINTS
   POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
       1   14156.00          2    9724.00          3      34.00          4      24.00
       5       4.98          6       4.98          7       5.00          8       4.90
       9       4.94         10       5.04         11       4.98         12       4.98
      13      28.00         14      26.00         15      14.00         16      18.00
      17     706.00         18     706.00         19     708.00         20     706.00
      21     706.00         22     704.00         23     716.00         24     708.00
      25     706.00         26     704.00         27     714.00         28     706.00
      29     704.00         30     716.00         31     704.00         32     712.00
      33     704.00         34     714.00         35     708.00         36     702.00
      37     712.00         38     704.00         39     716.00         40     704.00
 CLOSED LOOP POINTS
   POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
       1          0          2          0          3          0          4          0
       5          0          6          0          7         70          8         29

 ONE HOUR LOG     DAY  4      TIME     14.00

 NORMAL SCAN    DAY  4    TIME        14.00

 NORMAL SCAN    DAY  4    TIME        14.02
```

7

```
  ONE HOUR LOG    DAY  3     TIME      9.00


  NORMAL SCAN   DAY  3    TIME        9.00



     7    0    0    0    0    0    9    5
     0    0    0    0    0    0    0    0
  LOOP  7 NEW SET POINT  95 DAY 3 TIME   9.02


  NORMAL SCAN   DAY  3    TIME        9.02
  CE UNMASK CORE LOAD--PRESS START TO EXIT FROM CORE LOAD
  CEI 09.070
 SS0 ON ERR CNTRS
 SS1 ON 1443
 SS2 ON 2310
 SS3 ON 1053
 2310
 UNIT  USER  ORIGINAL
    1    00       01
    2    02       02
    3    00       00
 2310
 UNIT  USER  ORIGINAL
    1    01       01
    2    02       02
    3    00       00
 SWITCH PACKS


  NORMAL SCAN   DAY  3    TIME        9.04


  START OF GRADE     85 PRODUCTION TIME      0.20 START TIME     9.05 DAY  3
  OP-GUIDE LIMITS FOR NEW GRADE
  POINT    HIGH LIMIT     LOW LIMIT                POINT    HIGH LIMIT     LOW LIMIT
      1      32000.00     -32000.00                    2      32000.00     -32000.00
      3      32000.00     -32000.00                    4      32000.00     -32000.00
      5          5.50          4.50                    6          5.50          4.50
      7          5.50          4.50                    8          5.50          4.50
      9          5.50          4.50                   10          5.50          4.50
     11          5.50          4.50                   12          5.50          4.50
     13      32000.00     -32000.00                   14      32000.00     -32000.00
     15      32000.00     -32000.00                   16      32000.00     -32000.00
     17      32000.00     -32000.00                   18      32000.00     -32000.00
     19      32000.00     -32000.00                   20      32000.00     -32000.00
     21      32000.00     -32000.00                   22      32000.00     -32000.00
```



8

```
CLOSED LOOP POINTS
  POINT    VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
     1        0          2         0          3         0          4         0
     5        0          6         0          7        29          8        70


 ONE HOUR LOG     DAY   2      TIME      8.00


 START OF GRADE     95 PRODUCTION TIME      0.20 START TIME      8.04 DAY   2


 LOG15    DAY   2     TIME       8.15
 OP-GUIDE POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
     1    14154.00        2      9454.00        3        26.00        4       -62.00
     5        4.97        6         4.98        7         5.00        8         4.90
     9        4.94       10         5.04       11         4.98       12         4.98
    13        0.19       14         0.02       15         0.57       16        -0.00
    17      712.00       18       710.00       19       710.00       20       708.00
    21      710.00       22       710.00       23       710.00       24       712.00
    25      708.00       26       712.00       27       708.00       28       706.00
    29      710.00       30       708.00       31       706.00       32       710.00
    33      708.00       34       708.00       35       708.00       36       708.00
    37      708.00       38       706.00       39       710.00       40       710.00
 CLOSED LOOP POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
     1        0          2        -1          3         0          4         0
     5        0          6         0          7        69          8        30


 SHIFT END LOG    DAY   2      TIME      8.15


 START OF GRADE     96 PRODUCTION TIME      0.20 START TIME      8.25 DAY   2


 LOG15    DAY   2     TIME       8.30
 OP-GUIDE POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
     1    14154.00        2      9454.00        3        24.00        4       -80.00
     5        4.97        6         4.98        7         5.00        8         4.90
     9        4.94       10         5.04       11         4.98       12         4.98
    13        0.88       14         0.20       15         0.40       16        -0.00
    17      710.00       18       710.00       19       710.00       20       708.00
    21      716.00       22       712.00       23       710.00       24       710.00
    25      714.00       26       710.00       27       706.00       28       714.00
    29      708.00       30       708.00       31       712.00       32       714.00
    33      708.00       34       712.00       35       712.00       36       714.00
    37      706.00       38       710.00       39       712.00       40       712.00
 CLOSED LOOP POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
     1        0          2         0          3         0          4         0
     5        0          6         0          7        29          8        70


 MONDAY MORNING REPORT     DAY   2      TIME      8.30


 START OF GRADE     97 PRODUCTION TIME      0.20 START TIME      8.45 DAY   2


 LOG15    DAY   2     TIME       8.45
 OP-GUIDE POINTS
  POINT      VALUE      POINT      VALUE      POINT      VALUE      POINT      VALUE
     1    14156.00        2      9444.00        3        36.00        4       -82.00
     5        4.97        6         4.98        7         5.00        8         4.90
     9        4.94       10         5.04       11         4.98       12         4.98
    13        0.71       14         0.36       15         0.69       16        -0.00
    17      706.00       18       708.00       19       712.00       20       712.00
    21      712.00       22       716.00       23       710.00       24       712.00
    25      710.00       26       716.00       27       714.00       28       712.00
    29      710.00       30       714.00       31       710.00       32       710.00
    33      712.00       34       710.00       35       710.00       36       712.00
    37      710.00       38       710.00       39       710.00       40       710.00
 CLOSED LOOP POINTS
  POINT     VALUE      POINT     VALUE      POINT     VALUE      POINT     VALUE
     1        0          2         0          3         0          4         0
     5        0          6         0          7        36          8        63
```

8 (Continued)

```
LOG15   DAY  2    TIME      9.00
OP-GUIDE POINTS
  POINT      VALUE       POINT       VALUE       POINT       VALUE       POINT       VALUE
     1    14156.00           2     9438.00           3       22.00           4      -78.00
     5        4.97           6        4.98           7        5.00           8        4.90
     9        4.94          10        5.04          11        4.98          12        4.98
    13        0.28          14        0.36          15        0.67          16       -0.00
    17      708.00          18      714.00          19      708.00          20      710.00
    21      714.00          22      714.00          23      710.00          24      712.00
    25      712.00          26      710.00          27      712.00          28      714.00
    29      710.00          30      708.00          31      712.00          32      712.00
    33      708.00          34      708.00          35      710.00          36      712.00
    37      708.00          38      714.00          39      708.00          40      714.00
CLOSED  LOOP POINTS
  POINT      VALUE       POINT       VALUE       POINT       VALUE       POINT       VALUE
     1          0            2          -1           3          0            4          0
     5          0            6           0           7         69            8         30


ONE HOUR LOG     DAY  2      TIME      9.00
PRODUCTION STOP     TIME    9.10    DAY 2
```

9

```
// JOB        A
// *  UPDATE A PROCESS JOB FILE ON DISK
// XEQ LOADJ   FX

 PROCESS JOB FILE LOAD     TIME  11.14
 GRADE NUMBER 12345       PRODUCTION TIME       1.30
 OP-GUIDE LIMITS FOR GRADE
 POINT   HIGH LIMIT     LOW LIMIT                POINT   HIGH LIMIT     LOW LIMIT
     1     32000.00     -32000.00                    2     32000.00     -32000.00
     3     32000.00     -32000.00                    4     32000.00     -32000.00
     5         5.45          4.96                    6         5.45          4.96
     7         5.45          4.96                    8         5.45          4.96
     9         5.45          4.96                   10         5.45          4.96
    11         5.45          4.96                   12         5.45          4.96
    13     32000.00     -32000.00                   14     32000.00     -32000.00
    15     32000.00     -32000.00                   16     32000.00     -32000.00
    17     32000.00     -32000.00                   18     32000.00     -32000.00
    19     32000.00     -32000.00                   20     32000.00     -32000.00
    21     32000.00     -32000.00                   22     32000.00     -32000.00
    23     32000.00     -32000.00                   24     32000.00     -32000.00
    25     32000.00     -32000.00                   26     32000.00     -32000.00
    27     32000.00     -32000.00                   28     32000.00     -32000.00
    29     32000.00     -32000.00                   30     32000.00     -32000.00
    31     32000.00     -32000.00                   32     32000.00     -32000.00
    33     32000.00     -32000.00                   34     32000.00     -32000.00
    35     32000.00     -32000.00                   36     32000.00     -32000.00
    37     32000.00     -32000.00                   38     32000.00     -32000.00
    39     32000.00     -32000.00                   40     32000.00     -32000.00
 CLOSED LOOP SET POINTS FOR GRADE
   POINT      SETPT    POINT      SETPT      POINT      SETPT     POINT      SETPT
       1          0        2          0          3          0         4          0
       5          0        6          0          7         15         8         86

 PROCESS JOB FILE LOAD     TIME  11.14
 GRADE NUMBER  4530       PRODUCTION TIME       1.25
 OP-GUIDE LIMITS FOR GRADE
 POINT   HIGH LIMIT     LOW LIMIT                POINT   HIGH LIMIT     LOW LIMIT
     1     32000.00     -32000.00                    2     32000.00     -32000.00

     3     32000.00     -32000.00                    4     32000.00     -32000.00
     5         5.55          4.45                    6         5.45          4.45
     7         5.55          4.45                    8         5.45          4.45
     9         5.35          4.45                   10         5.65          4.45
    11         5.63          4.45                   12         5.62          4.45
    13     32000.00     -32000.00                   14     32000.00     -32000.00
    15     32000.00     -32000.00                   16     32000.00     -32000.00
    17     32000.00     -32000.00                   18     32000.00     -32000.00
    19     32000.00     -32000.00                   20     32000.00     -32000.00
    21     32000.00     -32000.00                   22     32000.00     -32000.00
    23     32000.00     -32000.00                   24     32000.00     -32000.00
    25     32000.00     -32000.00                   26     32000.00     -32000.00
    27     32000.00     -32000.00                   28     32000.00     -32000.00
    29     32000.00     -32000.00                   30     32000.00     -32000.00
    31     32000.00     -32000.00                   32     32000.00     -32000.00
    33     32000.00     -32000.00                   34     32000.00     -32000.00
    35     32000.00     -32000.00                   36     32000.00     -32000.00
    37     32000.00     -32000.00                   38     32000.00     -32000.00
    39     32000.00     -32000.00                   40     32000.00     -32000.00
 CLOSED LOOP SET POINTS FOR GRADE
   POINT      SETPT    POINT      SETPT      POINT      SETPT     POINT      SETPT
       1          0        2          0          3          0         4          0
       5          0        6          0          7         65         8         83
```

9 (Continued)

```
PROCESS JOB FILE LOAD     TIME  11.14
GRADE NUMBER  4531      PRODUCTION TIME       0.10
OP-GUIDE LIMITS FOR GRADE
POINT    HIGH LIMIT     LOW LIMIT              POINT    HIGH LIMIT     LOW LIMIT
    1      32000.00     -32000.00                  2      32000.00     -32000.00
    3      32000.00     -32000.00                  4      32000.00     -32000.00
    5          5.23          5.20                  6      32000.00     -32000.00
    7          5.23          5.21                  8      32000.00     -32000.00
    9      32000.00     -32000.00                 10      32000.00     -32000.00
   11      32000.00     -32000.00                 12      32000.00     -32000.00
   13      32000.00     -32000.00                 14      32000.00     -32000.00
   15      32000.00     -32000.00                 16      32000.00     -32000.00
   17      32000.00     -32000.00                 18      32000.00     -32000.00
   19      32000.00     -32000.00                 20      32000.00     -32000.00
   21      32000.00     -32000.00                 22      32000.00     -32000.00
   23      32000.00     -32000.00                 24      32000.00     -32000.00
   25      32000.00     -32000.00                 26      32000.00     -32000.00
   27      32000 00     -32000.00                 28      32000.00     -32000.00
   29      32000.00     -32000.00                 30      32000.00     -32000.00
   31      32000.00     -32000.00                 32      32000.00     -32000.00
   33      32000.00     -32000.00                 34      32000.00     -32000.00
   35      32000.00     -32000.00                 36      32000.00     -32000.00
   37      32000.00     -32000.00                 38      32000.00     -32000.00
   39      32000.00     -32000.00                 40      32000.00     -32000.00
CLOSED LOOP SET POINTS FOR GRADE
  POINT      SETPT      POINT      SETPT      POINT      SETPT      POINT      SETPT
     1          0          2          0          3          0          4          0
     5          0          6          0          7         45          8         47

PROCESS JOB FILE LOAD     TIME  11.14
GRADE NUMBER  4532      PRODUCTION TIME       5.45
OP-GUIDE LIMITS FOR GRADE
POINT    HIGH LIMIT     LOW LIMIT              POINT    HIGH LIMIT     LOW LIMIT
    1      14200.00      14100.00                  2       9890.00       9880.00
    3         30.00         20.00                  4        -10.00        -30.00
    5          5.10          4.90                  6          5.10          4.90
    7          5.10          4.90                  8          5.10          4.90
    9          5.10          4.90                 10          5.10          4.90
   11          5.10          4.90                 12          5.10          4.90
   13         25.00         15.00                 14         25.00         15.00
   15      32000.00     -32000.00                 16      32000.00     -32000.00
   17      32000.00     -32000.00                 18      32000.00     -32000.00
   19      32000.00     -32000.00                 20      32000.00     -32000.00
   21      32000.00     -32000.00                 22      32000.00     -32000.00
   23      32000.00     -32000.00                 24      32000.00     -32000.00
   25      32000.00     -32000.00                 26      32000.00     -32000.00
   27      32000.00     -32000.00                 28      32000.00     -32000.00
   29      32000.00     -32000.00                 30        720.00        700.00
   31        720.00        700.00                 32        720.00        700.00
   33      32000.00     -32000.00                 34      32000.00     -32000.00
   35      32000.00     -32000.00                 36      32000.00     -32000.00
   37      32000.00     -32000.00                 38        720.00        700.00
   39        720.00        700.00                 40        720.00        700.00
CLOSED LOOP SET POINTS FOR GRADE
  POINT      SETPT      POINT      SETPT      POINT      SETPT      POINT      SETPT
     1          0          2          0          3          0          4          0
     5          0          6          0          7         98          8         12

PROCESS JOB FILE LOAD     TIME  11.15

// JOB        A
// END OF ALL JOBS FOR NOW
```

APPENDIX A. TSX SYSTEM COMPOSITION AND CAPABILITIES

	ON-LINE SYSTEM		NONPROCESS MONITOR	OFF-LINE SYSTEM
	WITH TIME-SHARING	WITHOUT TIME-SHARING		
FIXED CORE CONTAINS	SKELETON	SKELETON	TASK	TASK
SKELETON AREA OF DISK CONTAINS	SKELETON	SKELETON	TASK	SKELETON
TSC IN FIXED CORE	YES	NO	NO	NO
NONPROCESS MONITOR (NPM) FUNCTIONS	YES	NO	YES	YES
CORE LOAD BUILD - NONPROCESS CORE LOADS	YES	NO	YES	YES
CORE LOAD BUILD - PROCESS CORE LOADS	YES	NO	NO	YES
RESULT OF COLD START	SYSTEM SKELETON	SYSTEM SKELETON	TASK	SYSTEM SKELETON
EXECUTE PROCESS CORE LOADS	YES	YES	NO	NO
EXECUTE PROCESS INTERRUPTS	YES	YES	NO	NO
EXECUTE PROGRAMMED INTERRUPTS	YES	YES	NO	NO
EXECUTE TIMER INTERRUPTS	YES	YES	NO	NO
DATA PROCESSING I/O FUNCTIONS	YES	YES	YES	YES
PROCESS I/O FUNCTIONS	YES	YES	YES	NO
NPM ERROR MESSAGES	YES	NO	YES	YES
TASK ERROR MESSAGES	NO	NO	YES	YES
EAC ERROR MESSAGES	YES	YES	NO	NO
CONTROL OF TASK FUNCTIONS	NO	NO	YES	YES
CONTROL OF NPM FUNCTIONS	TSC, NONPROCESS SUPERVISOR AND JOB STACK	NON-APPLICABLE	NONPROCESS SUPERVISOR AND JOB STACK	NONPROCESS SUPERVISOR AND JOB STACK
CONTROL OF PROCESS FUNCTIONS	SYSTEM SKELETON	SYSTEM SKELETON	NON-APPLICABLE	NON-APPLICABLE
METHOD OF CALLING CORE LOADS	NAME CARD, INTERRUPTS, CALLS TO: CHAIN, VIAQ, SPECL, BACK, DPART, INTEX, SHARE, //XEQ AND LINK	NAME CARD, INTERRUPTS, CALLS TO: CHAIN, VIAQ, SPECL, BACK, DPART, AND INTEX	//XEQ AND LINK	NON-APPLICABLE
VARIABLE AREA OF CORE CONTAINS	PROCESS CORE LOADS, NONPROCESS CORE LOADS, NONPROCESS MONITOR, EDP PROGRAMS, AND COLD START	PROCESS CORE LOADS, EDP PROGRAMS, AND COLD START	NONPROCESS MONITOR, SYSTEM LOADER, TASK UTILITIES, NONPROCESS CORE LOADS AND SKELETON BUILDER	NONPROCESS MONITOR, TASK UTILITIES, SYSTEM LOADER AND SKELETON BUILDER

APPENDIX B. SUMMARY OF TSX CALL STATEMENTS

Where Used Code	Statement	Description
I	CALL INTEX	Causes return of control to MIC on interrupt exit.
M	CALL CHAIN (NAME)	Mainline core load designated by NAME is loaded and executed.
M	CALL SPECL (NAME)	Mainline core load containing this call is saved on disk. Mainline core load designated by NAME is loaded and executed.
M	CALL BACK	Mainline core load saved as a result of a CALL SPECL is restored to core and execution continues with the statement following the special call.
M, I, N*, C	CALL QIFON (NAME, P, L, I, E)	Specified interrupts, that have been recorded, will be queued in the order called by the CALL QIFON statement and according to its designated parameters. NAME - name of the mainline core load. P - execution priority of the named mainline core load. L - interrupt level or indicator. I - PISW bit position indicator or CALL COUNT indicators. E - error parameter to specify the action to be taken if queue is full.
M, I, N*, C	CALL CLEAR (M, L, I, ..., L, I,)	Specified interrupts will be cleared of recorded status whether they were recorded or not. M specifies the number of L and I parameters to follow. L and I are the same as designated for CALL QIFON. If M = 0, all recorded status is cleared.
M, I, N*, C	CALL QUEUE (NAME, P, E)	Mainline core load designated by NAME is entered in core load queue with priority P and error option E.
M, I, N*, C	CALL UNQ (NAME, P)	Mainline core load designated by priority P and NAME will be removed from the core load queue.
M	CALL VIAQ	Last logical statement of a mainline core load. The first core load, of the highest priority entered in the queue, is loaded and executed.
M, I, C	CALL MASK (I, J)	Interrupt levels specified by data statements for I and J are masked (no unmasking occurs).

Where Used Code	Statement	Description
M, I, C	CALL UNMK (I, J)	Interrupt levels specified by data statements for I and J are unmasked (no masking occurs).
M, I, N*, C	CALL SAVMK (I, J)	Interrupt level mask status is saved in I and J (no masking or unmasking occurs).
M, I, C	CALL RESMK (I, J)	Interrupt levels are masked according to I and J (all others are unmasked).
M, I, N, C	CALL OPMON	Operation Monitor is reset.
M, I, N*, C	CALL TIMER (NAME, I, INT)	Interval timer specified by I (1 or 2) is set up to count INT intervals. After INT intervals have elapsed, ITC will branch to NAME (user's subprogram).
M, I, N*, C	CALL COUNT (IN, I, INB)	Program interval timer specified by I (1, 2, 3,...,9) is set to count INB intervals. Upon completion of INB intervals, the ITC will branch to the subroutine specified by IN (IN specifies a subroutine number from 0 - 31).
M, X	CALL SHARE (I)	The present core load is saved and nonprocess time-sharing is set up for I timed intervals.
I, C	CALL ENDTS	Time-sharing is terminated.
M, I, N*, C	CALL SETCL (I)	Programmed clock is set to equal I.
M, I, N*, C	CALL CLOCK (I)	Clock is read into I.
M, I, C	CALL LEVEL (L)	Calls the programmed interrupt specified by the hardware level L (L must be between 0 -23).
M, I, C	CALL DPART	Tests the operation level of present use and, if an interrupt level exists, executes a CALL INTEX; otherwise a CALL VIAQ is executed.

M	-	Mainline core loads only.
I	-	Interrupt core loads only.
N	-	Nonprocess core loads only.
C	-	Combination mainline and interrupt core load.
*	-	Must be an XEQ from core load area (INTERVAL TIMER CONTROL REQUIRED)
X	-	Must be an XEQ from core load area (TIME-SHARING REQUIRED)

This section describes the Assembler language equivalent of the FORTRAN CALL statements provided in the time-sharing executive system.

Machine Interval Timers

The Assembler language statements to call the TIMER subprogram are:

```
CALL TIMER
CALL NAME
DC   A
DC   B
```

where NAME is the name of the subprogram to be executed when the time specified by B has elapsed. A and B must be defined as:

A	DC	1	For machine interval timer (A).
		or	
		2	For machine interval timer (B).
B	DC	XX	The number of intervals to be counted before the subprogram is executed.

Programmed Interval Timers

The Assembler language statements to call the COUNT subprogram are:

```
CALL COUNT
DC   A
DC   B
DC   C
```

where the parameters A, B, and C must be defined as:

B	DC	1-9	Programmed timer number
C	DC	XX	The number of intervals to be counted before the subprogram is executed.
A	DC	0-31	Number of the subprogram to be executed when the time has elapsed.

The Assembler language statements to be used to read and to set the programmed real-time clock are:

Read:

```
CALL  CLOCK
DC    A
```

where A is the address of the location where the contents of the clock are to be stored.

Set:

```
CALL  SETCL
DC    A
```

where A must be defined as:

A	DC	XXXX	The time to be used for setting the clock. The time must be represented in hours and thousandths of hours (i.e., 00000 through 23999).

PSC Statements

The following Assembler language statements are equivalent to the FORTRAN language calls for core load sequencing.

```
CALL BACK  }
CALL ENDTS }  No parameters are required
CALL VIAQ  }  for these calls.
CALL DPART }
```

Call Chain:

```
CALL CHAIN
CALL NAME
```

where NAME is the name of the core load to be executed.

Call Special:

```
CALL SPECL
CALL NAME
```

where NAME is the name of the core load to be executed.

Call Queue:

```
CALL QUEUE
CALL NAME
DC    A
DC    B
```

where NAME is the name of a core load to be added to the queue. A and B must be defined as follows.

A	DC	1-32,767	Priority Number
B	DC	1-32,766	Replace the lowest priority entry on error condition
		or	
		0	Ignore the call on error condition
		or	
		32,767	Restart on error condition

Call Unqueue:

```
CALL UNQ
CALL NAME
DC    A
```

where NAME is the name of a core load whose entry is to be removed from the queue. A must be defined as follows:

A	DC	1-32,767	Priority Number

Call Time-Share:

```
CALL SHARE
DC    A
```

where A must be defined as follows:

A	DC	XX	Number of programmed timer base intervals to be used for nonprocess operations.

Call Programmed Settable Interrupts:

```
CALL LEVEL
DC    A
```

where A must be defined as:

A	DC	0-23	User specified hardware level to cause interrupt

Interrupt Calls

The following Assembler language statements are used to service and clear recorded interrupts.

Call Interrupt Exit:

CALL INTEX	No parameters are required for this call

Service Recorded Interrupts

```
CALL QIFON
CALL NAME
DC    A
DC    B
DC    C
DC    D
```

where NAME is the name of the core load to be serviced if recorded. A, B, C, and D must be defined as follows:

A	DC	XX	Priority number
B	DC	XX	Interrupt level number or indicator
C	DC	XX	Position within PISW or indicator
D	DC	1-32,767 or	Replace lowest priority entry on error condition
		0 or	Ignore the call on error condition
		32,767	Restart on error condition

Clear Recorded Interrupts:

```
CALL CLEAR        CALL CLEAR
DC   A            DC   A (when A = 0)
DC   B(1)
DC   C(1)
DC   B(2)
DC   C(2)
```

where A, B, and C must be defined as follows:

A	DC	XX	Number of Bs and Cs which follow. If zero, all recorded status is cleared
B	DC	XX	Interrupt level number or indicator. Not used if A = 0
C	DC	XX	Position within PISW or indicator. Not used if A = 0

Miscellaneous Subroutines:

The following Assembler language statements are used to link the miscellaneous subroutines.

Mask:

```
CALL  MASK
DC    A
DC    B
```

where A and B must be defined as:

A	DC	/0000	Levels to masked. A represents the first 14 levels (0 through 13). For example, to mask levels 0 - 13, use: /FFFC
B	DC	/0000	Levels to be masked. B represents the second 10 levels (14 through 23). For example, to mask levels 14 through 23, use: /FFC0

Unmask:

```
CALL  UNMK
DC    A
DC    B
```

where A and B must be defined the same as shown for CALL MASK. The designated levels are unmasked.

Save Mask:

```
CALL  SAVMK
DC    A
DC    B
```

where A and B are the addresses of the core storage words where the contents of the interrupt mask register are to be placed:

Restore Mask:

```
CALL  RESMK
DC    A
DC    B
```

where A and B are the levels defined for the CALL MASK or CALL UNMK.

Reset Operations Monitor:

CALL OPMON	No parameters are required for this call

Address		Description of Use
Decimal	Hexadecimal	
00000	0000	Reserved
00001	0001	Branch instruction (/4400)
00002	0002	CE routine entry address (EACA)
00003	0003	Not used
00004	0004	Interval timer A
00005	0005	Interval timer B
00006	0006	Interval timer C
00007	0007	General I/O busy indicator
00008	0008	Internal error interrupt branch address
00009	0009	Trace interrupt branch address
00010	000A	Mainline return address from CE routine
00011	000B	Level 0 interrupt address
00012	000C	Level 1 Interrupt address
00013	000D	Level 2 interrupt address
00014	000E	Level 3 interrupt address
00015	000F	Level 4 interrupt address
00016	0010	Level 5 interrupt address
00017	0011	Level 6 interrupt address
00018	0012	Level 7 interrupt address
00019	0013	Level 8 interrupt address
00020	0014	Level 9 interrupt address
00021	0015	Level 10 interrupt address
00022	0016	Level 11 interrupt address
00023	0017	Level 12 interrupt address
00024	0018	Level 13 interrupt address
00025	0019	Level 14 interrupt address
00026	001A	Level 15 interrupt address
00027	001B	Level 16 interrupt address
00028	001C	Level 17 interrupt address
00029	001D	Level 18 interrupt address
00030	001E	Level 19 interrupt address
00031	001F	Level 20 interrupt address
00032	0020	Level 21 interrupt address
00033	0021	Level 22 interrupt address
00034	0022	Level 23 interrupt address
00035	0023	1 = Loop on 1443 not ready during nonprocess program; 0 = go to EAC
00036	0024	Logical 1443-1 device table address
00037	0025	Logical 1443-2 device table address
00038	0026	Physical 1443-1 device table address
00039	0027	Physical 1443-2 device table address
00040	0028	Beginning address of MIC
00041	0029	User time-sharing time
00042	002A	Constant: -1
00043	002B	Constant: -10
00044	002C	Entry address to 1053 no response subroutine
00045	002D	Timer busy indicators
00046	002E	Mask register (0-13)
00047	002F	IOCC control word for UNMK1
00048	0030	Mask register (14-23)
00049	0031	IOCC control word for UNMK2
00050	0032	Mask levels 0-13
00051	0033	IOCC for levels 0-13
00052	0034	Mask levels 14-23
00053	0035	IOCC for levels 14-23
00054	0036	Pseudo accumulator (WK4)
00055	0037	Pseudo accumulator (WK5)
00056	0038	1 = time-sharing is in progress; 0 = not in progress
00057	0039	Address of magnetic tape sense control word
00058	003A	1 = ITC is in system director; 0 = ITC is not included
00059	003B	Non-zero indicates TASK is in core
00060	003C	Address of timer A sub-routine
00061	003D	Address of timer B sub-routine
00062	003E	Program timer 1
00063	003F	Timer 1 subprogram number (1-32)
00064	0040	Timer 1 on-off branch
00065	0041	Program timer 2
00066	0042	Timer 2 subprogram number (1-32)
00067	0043	Timer 2 on-off branch
00068	0044	Program timer 3
00069	0045	Timer 3 subprogram number (1-32)
00070	0046	Timer 3 on-off branch

Address		Description of Use
Decimal	Hexadecimal	
00071	0047	Program timer 4
00072	0048	Timer 4 subprogram number (1-32)
00073	0049	Timer 4 on-off branch
00074	004A	Program timer 5
00075	004B	Timer 5 subprogram number (1-32)
00076	004C	Timer 5 on-off branch
00077	004D	Program timer 6
00078	004E	Timer 6 subprogram number (1-32)
00079	004F	Timer 6 on-off branch
00080	0050	Program timer 7
00081	0051	Timer 7 subprogram number (1-32)
00082	0052	Timer 7 on-off branch
00083	0053	Program timer 8
00084	0054	Timer 8 subprogram number (1-32)
00085	0055	Timer 8 on-off branch
00086	0056	Program timer 9
00087	0057	Timer 9 subprogram number (1-32)
00088	0058	Timer 9 on-off branch
00089	0059	Timer-sharing timer
00090	005A	Exit address of I/O routines
00091	005B	Time-sharing timer busy indicator
00092	005C	Programmed clock
00093	005D	Programmed clock
00094	005E	Branch to ITC exit routine
00095	005F	Constant: -50
00096	0060	Constant: 3
00097	0061	PAUSE routine indicator specifying that interrupt has occurred
00098	0062	Entry address of I/O test routine (IOTST)
00099	0063	1 = CARDN is in skeleton
00100	0064	Mask routine indicator: 1 = out-of-core interrupts are masked
00101	0065	Address of CALL INTEX processing routines
00102	0066	Beginning address of variable core
00103	0067	TV location (XR3)
00104	0068	Interrupt level work area address (XR3)
00105	0069	Interrupt core load ending address
00106	006A	CALL CHAIN entry
00107	006B	System director ending address
00108	006C	Mask word out-of-core interrupts (0-13)
00109	006D	Mask word for out-of-core interrupts (14-23)
00110	006E	System mask save area (0-13)
00111	006F	System mask save area (14-23)
00112	0070	1 = AI basic overlap feature available
00113	0071	1 = AI expander overlap feature available
00114	0072	Entry address for I/O error
00115	0073	EAC error code
00116	0074	Error information
00117	0075	Error information
00118	0076	Error information
00119	0077	Error information
00120	0078	Entry address of EAC
00121	0079	Address of queue table
00122	007A	Maximum number of queue entries
00123	007B	Number of interrupt levels used (NULEV)
00124	007C	Entry address of disk routine
00125	007D	Entry address of list printer routine
00126	007E	Entry address of system printer routine
00127	007F	Constant: /0600
00128	0080	Constant: /0500
00129	0081	Constant: /F800
00130	0082	Constant: /0FF8
00131	0083	Constant: /00FF
00132	0084	Constant: /8000
00133	0085	Constant: /0001
00134	0086	Constant: /0002
00135	0087	Constant: /0004
00136	0088	Constant: /0005
00137	0089	Constant: /0007
00138	008A	Constant: /0FFF
00139	008B	Constant: /2000
00140	008C	Constant: /0180

Address		Description of Use	Address		Description of Use
Decimal	Hexadecimal		Decimal	Hexadecimal	
00141	008D	Constant: 320	00171	00AB	TSC indicator: 0 = Call nonprocess monitor; 1 = Call program from save area
00142	008E	CALL LINK entry address	00172	00AC	Entry point to TVSAV
00143	008F	Address of EAC disk down message	00173	00AD	Entry point to TVEXT
00144	0090	Constant: 321	00174	00AE	Ending address of skeleton I/O area
00145	0091	Address of first word after ETV	00175	00AF	Address of 1053 logical table
00146	0092	Constant: /FF00	00176	00B0	Address of message buffer table
00147	0093	Constant: /F000	00177	00B1	Not used in an on-line system. Under TASK, this is the starting address of the TASK core dump program.
00148	0094	Constant: /FF87	00178	00B2	Address of EAC constants
00149	0095	Keyboard request indicator	00179	00B3	Entry address of nonprocess monitor read-in routine
00150	0096	Program timer busy indicator	00180	00B4	Non-disk FIO save area address
00151	0097	Not used	00181	00B5	Disk FIO save area address
00152	0098	Entry address to set timers busy	00182	00B6	Entry point to EXIT subroutine
00153	0099	Entry point to EAC printer routine	00183	00B7	Address of message buffer disk address
00154	009A	Entry point to QZSAV	00184	00B8	Entry for DISKN from ETV
00155	009B	Entry point to QZEXT	00185	00B9	Entry for TYPEN from ETV
00156	009C	Starting address of skeleton COMMON	00186	00BA	Entry for PRNTN from ETV
00157	009D	Length of skeleton COMMON	00187	00BB	Address of disk drive table
00158	009E	Address of message buffer drive code	00188	00BC	Address of logical drive 0 device table
00159	009F	I/O Error routine entry/ return address	00189	00BD	Address of logical drive 1 device table
00160	00A0	Program interrupt IOCC (0-13)	00190	00BE	Address of logical drive 2 device table
00161	00A1	IOCC control word	00191	00BF	Address of physical drive 0 device table
00162	00A2	Program interrupt IOCC (14-23)	00192	00C0	Address of physical drive 1 device table
00163	00A3	IOCC control word	00193	00C1	Address of physical drive 2 device table
00164	00A4	TASK nonprocess monitor abort indicator	00194	00C2	Address of save area for unformatted FIO
00165	00A5	EAC printer type code: 0 = 1053, 1 = 1443	00195	00C3	Address of inskel ETV
00166	00A6	List printer type code: 0 = 1053, 1 = 1443	00196	00C4	Address of variable ETV
00167	00A7	System printer type code: 0 = 1053, 1 = 1443	00197	00C5	Special save indicator
00168	00A8	Core size minus 1	00198	00C6	User time-sharing (TISHA) for CALL VIAQ when Queue is empty.
00169	00A9	Address of 1442 entry in interrupt branch table			
00170	00AA	Address of FORTRAN I/O table			

INDEX

READER'S COMMENT FORM

IBM 1800 Time-Sharing Executive System
Concepts and Techniques

Form C26-3703-0

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

	Yes	No
• Does this publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? ________________________
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other ________________________

- Please give specific page and line references with your comments when appropriate.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

C26-3703-0

YOUR COMMENTS, PLEASE...

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. 232

fold fold

IBM 1800 Printed in U.S.A. C26-3703-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

C26-3703-0

IBM 1800 Printed in U.S.A. C26-3703-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]