COMPAGNIE INTERNATIONALE POUR L'INFORMATIQUE

10070 BATCH PROCESSING MONITOR

user's manual

C900954 B/En

10 070 Computer
C900954 B/En
Price : 60,00 FF

10070 BATCH PROCESSING MONITOR

user's manual

COMPAGNIE INTERNATIONALE POUR L'INFORMATIQUE

REVISION

This publication, C 900 954 B/En, is a revision of the 10 070 Batch Processing Monitor user's manual C 900 954 A/En dated February 1968. A change in text from that of the previous manual is indicated by a vertical line in the margin of the page.

RELATED PUBLICATIONS

TITLE	PUBLICATION N°
10 070 Mathematical Routines User's Manual	C 900 906
10 070 Computer Description Manual	C 900 950
10 070 SYMBOL and METASYMBOL User's Manual	C 900 952
10 070 Basic Control Monitor User's Manual	C 900 953
10 070 Stand-Alone Systems Operations Manual	C 901 053
10 070 FORTRAN IV User's Manual	C 900 956
10 070 FORTRAN IV-H User's Manuel	C 900 966
10 070 FORTRAN IV-H Operations Manual	C 901 144
10 070 RAD SORT User's Manual	C 901 199
10 070 Batch Processing Monitor Operations Manual	C 901 198
10 070 COBOL-65 User's Manual	C 901 500

NOTICE

The specification of the software system described in this publication are subject to change without notice. The availibility or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory.

CONTENTS

APPENDIXES

ILLUSTRATIONS

TABLES

DEFINITION OF TERMS

absolute loader: a resident Monitor routine capable of loading a program into a core storage area beginning with the location specified in the first record of the program.

addend value: a hexadecimal constant to be added to the value of a relocatable address. The constant is expressed as a signed integer appended to the address; e.g., START+12 or HERE-F1.

address resolution code: a 2-bit code that specifies whether an associated address is to be used as a byte address or is to be converted (by truncating low order bits) to a halfword, word or doubleword address.

background area: that area of core storage allocated to batch processing. Programs executed in this area may be interrupted by foreground programs.

background program: any program executed under Monitor control in the background area of core storage.

binary input: input from the device to which the BI (binary input) operational label is assigned.

checkpointed job: a partially processed background job that has been saved in secondary storage along with all registers and other "environment" so that the job can be restarted.

common page: a page of core storage that is available to the user's program and in which stored data is retained until the current job is terminated or until the page is released by the user's program.

console interpreter: a Monitor routine that interprets graphic and control codes input from a keyboard (TY device).

control command interpreter: a Monitor routine that interprets control commands.

control command: any control message other than a key-in. A control command may be input via any device to which the system command input function has been assigned (normally a card reader).

control function: any Monitor function initiated by a control command or control key-in.

control key-in: a control message of the type that must be input from the operator's console.

control message: any message received by the Monitor that is either a control command or a control key-in (see "key-in").

cooperative: a Monitor routine that transfers information between a user's program and disc storage (also see "symbiont").

Data Control Block (DCB): a table that contains the information used by the Monitor in the performance of an I/O operation.

declaration: a load item that introduces a symbolic name, so that the loader can give it a unique name number.

declaration number: the name number given to the symbolic name associated with a particular declaration.

dedicated memory: core memory locations reserved by the Monitor for special purposes, such as traps, interrupts, and real-time programs.

definition: a load item that assigns a specific value to the symbolic name associated with a particular name number.

dummy section: a type of program section that provides a means by which more than one subroutine may reference the same data (via an external definition used as a label for the dummy section).

element file: a user's file consisting of program elements, such as relocatable object modules or library load modules.

end record: the last record to be loaded, in an object module or load module.

error card: a card punched by the Monitor following each card punched in error on a single-stacker punch. The letters ERR are punched on each error card.

error severity level code: a 4-bit code indicating the severity of errors noted by the processor. This code is contained in the final byte of an object module.

Event Control Block (ECB): a word containing flags set by the Monitor to indicate the completion of events such as segment loading or solicited key-ins.

expression: a series of load items immediately preceded by an "origin", "define field", "forward reference definition", "external definition", or "define start" load item and terminated by an "expression end" load item (see Appendix A).

expression accumulator: a register in which the object module loader stores values generated during the evaluation of an expression.

external definition: a load item that assigns a specific value to the symbolic name associated with a particular external definition name number. An external definition allows the specified symbolic name to be used in external references (see below).

external reference: a reference to a declared symbolic name that is not defined within the object module in which the reference occurs. An external reference can be satisfied only if the referenced name is defined by an external load item in another object module.

file management routines: Monitor routines that interpret and perform I/O functions.

foreground program: a real-time program executed on the occurrence of a priority interrupt signal or an unsolicited key-in.

forward reference number: a number assigned by a processor to designate a specific forward reference in a source program.

Function Parameter Table (FPT): a table through which a user's program communicates with a Monitor function (such as an I/O function).

GO file: a temporary disc file of relocatable object modules formed by a processor. Such modules may be retrieved by the use of a LOAD control command.

granule: a block of disc sectors large enought to contain 512 words of stored information.

idle state: the state of the Monitor when it is first loaded into core memory or after encountering a FIN control command. The idle state is ended by means of an S key-in.

installation control command: any control command used during System Generation to direct the formatting of a Monitor system.

I/O supervisor: a Monitor routine that controls the queue of outstanding I/O requests.

Job Information Table (JIT): a table in which the Monitor retains information that is needed for handling the current job (e.g., memory area allocated, SPD address, etc.).

key-in: information entered by the operator via a keyboard.

keyword: a word, consisting of from 1 to 8 characters, that identifies a particular operand used in a control command.

library input: input from the device to which the LI (library input) operational label is assigned.

library load module: a load module that may be combined (by the relocating loader) with relocatable object modules, or other library load modules, to form a new executable load module.

load bias: a value replacing the origin of a relocatable program, to change the address at which program loading is to begin.

load information: information (i.e., control information, data, and instructions) generated by a processor and contained in one or more modules capable of being linked to form an executable program.

load item: a load control byte followed by any additional bytes of load information pertaining to the function specified by the control byte.

load location counter: a counter established and maintained by the Monitor to contain the address of the next location into which information is to be loaded.

load map: a listing of significant information pertaining to the storage locations used by a program (see Appendix L).

load module: an executable program formed by the relocating loader, using relocatable object modules and/or library load modules as source information.

logical device: a peripheral device that is represented in a program by an operational label (e.g., BI or PO) rather than by a specific physical device name.

monitor: a program that supervises the processing, loading, and execution of other programs.

name number: a number assigned by the relocating loader to identify a declared name.

object deck: a card deck comprising one or more object modules of load information.

object language: the arbitrary, conventional, binary language in which the output of a processor is expressed.

object module: the series of records containing the load information pertaining to a single program or subprogram (i.e., from the beginning to the end). Object modules serve as input to the relocating loader.

operational label: a symbolic name used to identify a logical system device.

option: an elective operand in a control command or procedure call, or an elective parameter in a Function Parameter Table.

overlay loader: a Monitor routine that loads and links elements of overlay programs.

overlay program: a segmented program in which the element (i.e., segment) currently being executed may overlay the core storage area occupied by a previously executed element.

parameter presence indicator: a bit, in word 1 of a Function Parameter Table that indicates whether a particular parameter word is present in the remainder of the table.

physical device: a peripheral device that is referred to by a "name" specifying the device type, I/O channel, and device number (also see "logical device").

postmortem dump: a listing of the contents of a specified area of core memory, usually following the abortive execution of a program.

primary reference: an external reference that must be satisfied by a corresponding external definition (capable of causing loading from the system library).

Program Trap Conditions (PTC): two words that indicate trap status (set or reset) and trap exit address, respectively.

pseudo file name: a symbolic name used to identify a logical device in a user's program.

public library: a set of library routines declared, at System Generation time, to be public (i.e., always resident during execution of a user's program).

relocatable object module: a program, or subprogram, generated by a processor such as Meta-Symbol, FORTRAN, COBOL, etc. (in 10 070 object language).

relocating loader: a program capable of loading one or more object modules and linking them to form an executable program.

resident program: a program that has been loaded into a dedicated area of core memory.

scheduler: a Monitor routine that controls the initiation and termination of all jobs.

secondary reference: an external reference that may or may not be satisfied by a corresponding external definition (not capable of causing loading from the system library).

secondary storage: any rapid-access storage medium other than core memory (e.g., magnetic disc).

segment loader: a Monitor routine that loads overlay segments from disc storage at execution time.

source deck: a card deck comprising a complete program or subprogram, in symbolic format.

source language: a language used to prepare a source program (and therefrom a source deck) suitable for processing by an assembler or compiler.

standard control section: a control section whose length is not known by a 1-pass processor until all the load information for that section has been generated.

symbiont: a Monitor routine that transfers information between disc storage and a peripheral device (also see "cooperative").

symbolic input: input from the device to which the SI (symbolic input) operational label is assigned.

symbolic name: an identifier that is associated with some particular source program statement or item so that symbolic references may be made to it even though its value may be subject to redefinition.

system library: a group of standard routines in object-language format, any of which may be incorporated in a program being formed.

system register: a register used by the Monitor to communicate information that may be of use to the user's program (e.g., error codes). System registers SR1, SR2, SR3, and SR4 are current general registers 8. 9, 10, and 11, respectively.

Task Control Block (TCB): a table of task control information set up and maintained by the Monitor. Among other things, it contains the data required to allow reentry of library routines during program execution.

TSS temp stack: a push-down stack established by the Monitor for use by an executing program (unless NOTCB was specified for the load module).

wait state: the state of the Monitor when it has encountered a W key-in, during which all backgroun operations are suspended until the state is ended by means of an E, S, or X key-in.

1. INTRODUCTION

OPERATING SYSTEM

The 10 070 Batch Processing Monitor functions as the major control element in an installation's operating system. The operating system consists of the Monitor and a number of processing programs: language translators, service programs, batch user's programs and real-time user's programs. In general the Monitor governs the order in which these programs are executed and provides common services to all of them (see Figure 1).

The number, types and versions of the programs in an operating system vary, depending upon the exact requirements at a particular installation. Each operating system consists of a selection of Monitor routines and processing programs that are closely integrated for a given range of applications.

The operating system required for a particular installation is generated through use of the 10 070 Batch Processing Monitor System Generation program.

As the requirements of an installation increase, the operating system can easily be enlarged, modified, or updated. The ability to adapt conveniently to new requirements is inherent in the system design. Once a system is generated, it can quickly be expanded to include user's programs, data, and system libraries. User's programs and the standard system processors are equivalent in that they are stored, cataloged, and referred to within the system in the same way. They are also written using the same conventions for communicating with the Monitor.

A user's program and/or data may be incorporated in the operating system temporarily or it may remain a part of the system for an extended period of time.

The operating system is self-contained and requires operator intervention only under exceptional conditions.

Operating procedures are given in the 10 070 Batch Processing Monitor Operations Manual (C 901 198).

PHILOSOPHY OF OPERATION

The Monitor uses sophisticated techniques for efficient machine operation in a production environment, while offering concurrent, critical real-time processing on a dynamic and compatible basis. The full multiuse capability of the Monitor provides for three levels of operation:

1. Batch processing.
2. Peripheral processing (symbionts): card-to-disc, disc-to-punch, disc-to-printer.
3. Real-time foreground processing.

BATCH PROCESSING

The ability to process a continuous series of jobs with little or no operator intervention is one of the most important features of the system. By reducing the need for operator participation, the operating system ensures faster throughput, and operations are less subject to error. For the most part, the operator should only have to perform routine tasks such as loading and unloading tape reels.

Complete and easy-to-use I/O services are available to user programs, thus relieving the programmer of many coding chores. Device assignment is general and automatic,

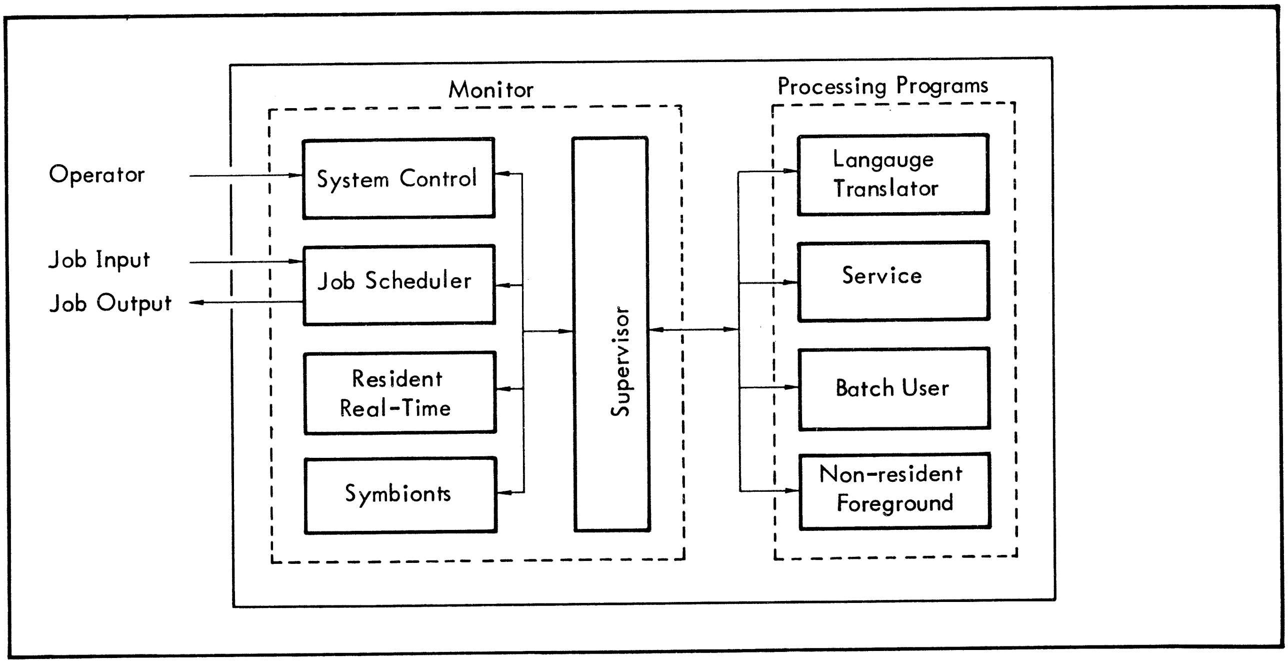


Figure 1. Operating System

enabling the user's program to exploit the complete flexibility of 10 070 Peripheral units.

I/O service is comprehensively organized to simplify programming and make machine utilization efficient. I/O transfers are automatically buffered, and I/O peripherals are serviced on a queue basis (by job). Jobs can thus be executed sequentially even though they might normally be I/O-bound and delay use of the CPU or other I/O devices.

Jobs can be run from a card-reader stack in the simplest case, or run under the rules of an installation's job scheduler by using the card-to-disc symbiont. The job scheduler permits selective job operation based on job type or administrative priority to maximize throughput efficiency or environmental needs. The computer operator maintains complete control over the job stack on secondary storage. Jobs can be suspended, initiated on a priority basis, or reorganized in sequence.

Secondary storage (disc) management is essential to efficient operation of the Monitor, since such storage is fully exploited in various ways. It is used for system storage to overlay portions of the Monitor, minimizing core memory residency. Service processors (compilers, assemblers, etc.) are contained on the disc for immediate access, and they too capitalize on rapid overlay techniques to minimize core memory requirements at execution time. Scratch storage for service processors and user programs is available on the disc, as is check-point storage. Finally, the secondary storage accommodates permanent and temporary user files.

Because of the Monitor's extensive use of high-speed secondary storage, comprehensive facilities for efficient storage allocation and access are provided by a Disc Management Routine. A choice of random or sequential access to files is provided to operating programs. Security features are offered to users so that assigned storage areas can be designated as public or private, read-only, or read-and-write. Private files are given a general password mechanism whereby access can be designated in various ways according to the needs of the installation.

User programs can avail themselves of the rapid-access disc and overlay service of the Monitor. With these facilities, user programs that require more operating core memory storage than is physically available can be easily segmented and controlled so that only part occupies available core memory at any one time. The Monitor accepts the overlay structure of the user's program and ensures proper sequencing and transferring of program elements. It also detects inconsistencies in the logical overlay structure and logs them as a diagnostic message to the user.

"Check-point" service can be called either by the user program itself or a real-time foreground process. In all cases, the user's program is saved on secondary storage for reactivation at the checkpoint, when appropriate. Thus, for a long production task, the program can call for periodic check-pointing. If a malfunction occurs, data files are repositioned so that the program can then continue from the last checkpoint. Finally, it may be necessary to checkpoint the background process if a real-time foreground task requires additional temporary core memory.

The Monitor provides for complete accounting user job activity on the 10 070 computer. Because of the system's multiusage capability, the accounting information indicates both elapsed time and actual machine facility utilization of each job.

The Monitor's loader function relocates user programs into the currently available core memory space, satisfies all library subroutine references, and links all program elements called for by the user. In addition, run-time debugging calls are recognized and established for the binary programs.

PERIPHERAL PROCESSING

Peripheral processing tasks (symbionts) are primarily I/O functions that require minimum service from a central processor (CPU); such processing can be performed concurrently with other computing tasks and other I/O processing. Symbionts consist of small routines that control I/O transactions and buffer storage in core memory to accommodate the data throughput. With a minimum core memory, high-speed I/O transactions can be processed without interfering with other concurrent processing tasks.

The Monitor permits the computer operator to directly call, or a user program to indirectly call, for the initiation of a symbiont. The symbionts provided are established in core memory as extensions to the operating system, and they communicate directly with the operator for termination of error conditions.

Because symbiont operation may be independent of and concurrent with other system actions, computer efficiency is maximized by overlapping bulk I/O transactions with normal processing. In particular, the rapid-access secondary storage philosophy of 10 070 operating systems capitalizes on the use of symbionts to derive maximum efficiency from the various service processors such as compilers and assemblers.

REAL-TIME PROCESSING

Real time processing, the most critical aspect of multiusage, involves processes that must react to external events (including clock pulses) within milliseconds or microseconds. Such tasks must be given priority treatment in varied ways. Although geared to job-shop operations, the Monitor permits 10 070 hardware to process real-time tasks and background tasks concurrently.

Real-time processes can be installed permanently as extensions of the resident Monitor or they can be dynamically loaded and initiated. The first method is used when the real-time process normally remains unchanged and is constantly operative. The dynamic approach can be used when real-time operations are executed periodically or irregularly, as in an experimental laboratory.

At installation time, a real-time process is assigned machine facilities on a dedicated basis. These facilities include disc and core memory residency, I/O channels, peripheral devices, external interrupt lines, and CAL trap locations. Such allocation remains in force until either the process or the computer operator terminates the program.

Finally, the real-time process can call for a secondary service that employs normal system facilities (nondedicated) without disturbing normal operations. Such a task might be an I/O transfer of accumulated data or the transmittal of a message to the operator.

If the real-time process requires use of the input/output processor, the Monitor recognizes I/O interrupts for the process by means of the task's priority level (hardware assignment) and immediately turns control over to the responsible program. Thus, although I/O interrupts are centralized, a real-time process still retains control over all its operations.

For real-time processes requiring operation on a periodic time basis, the Monitor offers a clock-watching service. This facility activates the process when the appropriate time elapses (within the precision of the standard clock). The user need not provide special clocks for the private use of a real-time program if the standard clock facility is adequate.

The Monitor can service several independent real-time processes concurrently. The user must be judicious in the allocation of time resources; however, to eliminate conflict in processing time between the various real-time processes, hardware assignment dictates priority. With proper planning, machine resources can be fully utilized without overloading the system.

PROCESSING PROGRAMS

A wide variety of processing programs are available for inclusion in the operation system. These may be supplemented by others supplied by the user or by CII . These programs are designed to increase throughput, decrease the response time of the system, enable a flexible and orderly growth of the system through a broad range of applications, and assist in programming.

LANGUAGE TRANSLATORS

The language translators assist a user by enabling him to define a program in a language form that can be readily learned and understood. The language translators provided by the system are discussed below.

FORTRAN

Compilers can be included in the operating system for compiling programs written in the FORTRAN IV language.

COBOL

A compiler may be included in the operating system for compiling programs written in the COBOL 65 language.

SYMBOL

A Symbol assembler may be included in the operating system for assembling programs written in the Symbol language. Symbol is a 1-pass assembler that provides for literals, forward references, and external definitions.

META-SYMBOL

A Meta-Symbol meta-assembler may be included in the operating system for assembling programs written in the Meta-Symbol language. Meta-Symbol is a multi-pass, high-level language, assembler. Its features compare favorably with those of the most advanced operational assemblers on the largest computing systems. Meta-Symbol is a superset of the Symbol language.

A comprehensive set of system procedures is provided for communicating service requests to the Monitor supervior.

SERVICE PROGRAMS

Service programs assist the programmer by providing routines for performing frequently used functions. The service programs consist of a Loader, Sort routine, relocatable library, and a set of utility programs (e.g., file management and media conversion).

LOADER

A Loader is provided for loading the output from a compiler or assembler into memory. The loader can also select any routines that are required from object module and load module libraries, and effectively combine them to form a single program, ready to be loaded into memory and executed.

The Loader enables changes to be made in a program without recompiling or reassembling it. In addition, it provides a means by which the program can be divided, if it is too large for the space available in memory (i.e., executed segments of the program can be overlaid by segments yet to be executed).

SORT

Sort is a generalized program for use in sorting fixed-length records. The sorting uses a direct-access storage device to minimize sort time, and the program takes full advantage of the input/output resources allocated by the Monitor.

UTILITY PROGRAMS

A set of utility programs is provided for performing functions such as:

Transferring data from one storage device or input/output device to another

Listing an inventory of data and programs that are cataloged

Listing files

Tape positioning and general control

MONITOR

The Monitor is a disc-oriented multiprogramming system designed to ensure efficient facility operation. It controls and coordinates the processing of a continuous series of batch and real-time jobs, and controls the I/O operations.

The Monitor enables users to fully exploit the capabilities of the CII 10 070 computer system by providing the computing power and flexibility usually found only on larger, more expensive systems.

The Monitor system is a control and production tool the user can adapt to a variety of environmental requirements. It simplifies computer utilization in a job-shop environment, where maximum efficiency is required on a production basis. In addition, the system can accommodate real-time foreground processing on a dynamic basis; that is, critical real-time tasks can be either permanently embedded in the operating environment or established and dismissed under operator control. Under the fundamental concept of multiusage, foreground tasks can be processed compatibly and concurrently with both a background production job stack and peripheral processors (symbionts), thus realizing three levels of concurrent processing.

A batch job is the basic independent task performed by the operating system. Each such background job is independent of any other job and consists of one or more directly or indirectly related job steps. A job step results in the execution of a processing program such as a language translator, loader, utility service, or user's program.

Job steps may be related to one another; the output of one may be passed on as the input to another.

Parts of the Monitor must remain resident to ensure continuous coordinated operation. Other parts are brought into core memory from secondary storage as they are required to perform specific functions.

Secondary storage management is essential in the Monitor. It is used for system storage to overlay portions of the Monitor, thus minimizing core memory residency. Processing programs are retrieved from disc and they too capitalize on rapid overlay techniques to minimize core memory requirements.

Scratch storage for service processors and user's programs is available on the disc, as is checkpoint and symbiont storage. In addition, the secondary storage accommodates permanent and temporary user's files.

Security features are offered to the user, so that assigned storage areas can be designated as public or private, read only, or read and write. A general password mechanism is provided whereby access can be designated in various ways according to the needs of the user's installation.

The Monitor provides a comprehensive group of facilities that feature automatic and efficient control of many data processing operations previously performed by the user.

The file management cataloging scheme enables the user to retrieve data and programs (including loadable programs) by symbolic name alone. Thus, the user is freed from the necessity of maintaining inventory lists of data and programs stored within the system. Cataloging reduces manual intervention and human error, and provides for protection, security, and sharing of files. Control of confidential data is provided by the system, and a user may prevent unauthorized access to a file. A secure file is made available for processing only when the correct password is furnished.

The file access facilities provided by the Monitor are a major expansion of the I/O control systems of previous operating systems. I/O routines are provided to efficiently schedule and control the transfer of data between main storage and I/O devices.

File management provides for

Reading and writing data

Blocking and deblocking records

Overlapping I/O and processing operation

Reading, verifying and writing file labels

Automatic file volume positioning

User error and abnormal condition checking

User label checking

Permitting the user to store, modify, and retrieve programs and data by symbolic name

Freeing the user from concern about specific I/O device configurations

Making device assignment external to the user's program, and

Sharing user's system files with

Complete file security

BPM features are summarized as follows:

Efficient and comprehensive I/O service to user programs

Automatic job sequencing for closed-shop operations

Dynamic real-time process initiation and execution

Maximum utilization and comprehensive management of rapid-access secondary storage (disc)

Comprehensive control of system operation by computer operator

Sophisticated (but easy-to-use) processor services for program creation, debugging, and execution such as FORTRAN IV, and Meta-Symbol

Recognition of administrative priority assignment on incoming jobs

Checkpoint service

Automatic job accounting

- Flexible job scheduling for efficient throughput operation and recognition of installation priorities
- Unimplemented instruction traps for simulation of unavailable hardware options
- Sophisticated error processing
- Modular, flexible design for user modification
- User of overlay techniques to minimize core memory residency
- Upward compatibility with the Universal Time-Sharing Monitor
- Tape Label processing
- Scheduling and controlling of I/O operations
- Blocking of disc and tape files
- Priority scheduling
- General and automatic device assignment
- Complete memory protection of the operating environment and real-time processes
- Comprehensive secondary storage management

GENERAL ORGANIZATION

SYSTEM CONTROL

The primary function of system control is to provide a two-way communication link between the operator and the system. The operator may command the system to change the status of a device, alter the operation of the system, and request status information. He is also responsible for alerting the job scheduler to initiate the reading and processing of jobs and rescheduling the order in which they are to be processed.

JOB SCHEDULER

The job scheduler is called into memory to prepare each job to be run. It performs its functions between jobs and between job steps but is not resident while a processing program is executing. The job scheduler is initiated by an operator key-in and directed by programmer-supplied control commands. The control command input stream is under complete control of the job scheduler; it performs various functions based on the control commands processed.

There are two versions of the job scheduler. One of these reads and processes jobs sequentially. The other (for the symbiont system) can read jobs concurrently from several input devices, such as card readers, and output on several output devices, such as printers, card punch, typewriter, and paper tape. Reading and writing can occur either independently of, or concurrently with, the actual processing of the jobs.

SUPERVISOR

The supervisor is the heart of the operating system. Its primary function is to provide a variety of services for other parts of the system while coordinating and controling the performance of these services to ensure efficient use of the facilities and resources. It also prevents programs from interfering with one another and with the operation of the Monitor. This is accomplished by its use of privileged instructions, such as storage-protection and I/O.

The supervisor receives control by means of program interruption. This may be caused by a specific request for services or it may be an automatic interruption, such as an I/O interrupt.

The I/O interrupts, in general, enable the supervisor to coordinate and effectively maintain control over the physical and programming resources of the system.

Services performed by the supervisor may be the result of a specific request, such as a request for storage space, or it may be a service that is virtually automatic, such as attempting to recover from an error condition.

JOB ORGANIZATION

The user controls the construction and execution of his program by means of control cards placed before, within, and following the input card decks. These control cards, interpreted by the Monitor, specify

- Processors required and the options to be used
- Input/output devices required and their specific assignments
- Loading and execution requirements
- Libraries and supporting services required
- Program modification and debugging requirements

Monitor routines may be used to handle all I/O functions and to perform other services for the user's program. To provide linkages between the user's program and a Monitor routine, an appropriate calling sequence must be present in the user's program when it is executed. Such a calling sequence can be generated at assembly time through the use of a Meta-Symbol procedure call in the source program.

If the Symbol assembler is used instead of the Meta-Symbol assembler, the calling sequence must be included explicitly in the user's source program (see Chapter 4, "System Procedures"), since the Symbol assembler does not process procedure calls. Compilers such as CII 10 070 FORTRAN IV generate all necessary calling sequences automatically.

Jobs may be input directly from a card-reader stack or under control of the system's job scheduler, through the use of a card-to-disc symbiont. Use of the scheduler permits selective job operation based on job priority. At all times, the operator may exercise complete control over the job queue in secondary storage. Jobs may be suspended, initiated on a priority basis, or reorganized in sequence.

Resident foreground tasks may be created and included in the foreground element file when the Monitor system is generated.

Nonresident foreground tasks may be created during normal batch operation. Both resident and nonresident foreground tasks may be updated during batch operation.

The Monitor can service several independent real-time processes concurrently. However, the user must be judicious in the allocation of time resources. To preclude conflicts between real-time processes, relative priorities are established uniquely by hardware interrupt assignments.

A foreground task may cause the background process to be checkpointed if additional core storage area is required for the real-time program. The Monitor's checkpoint routines are reenterable, and all data needed to restart a checkpointed job is saved along with the job.

For real-time processes requiring periodic operation, the Monitor offers a clockwatching service. This facility activates the real-time process when a specified time period has elapsed.

MONITOR ORGANIZATION

The Monitor system includes the following programs:

1. Control command interpreter (partly resident)
2. Console interpreter (partly resident)
3. Scheduler (nonresident)
4. File management routine (resident and nonresident)
5. Language processors (nonresident)
6. Relocating loader (nonresident)
7. Overlay loader (nonresident)
8. Segment loaders (resident and nonresident)
9. User service routines (resident and nonresident)
10. Foreground programs (resident and nonresident)
11. Library routines (nonresident)
12. Debug routines (nonresident)
13. I/O supervisor (resident)
14. I/O control routines (resident)
15. Cooperative and symbionts (partly resident)

The control command interpreter reads all control commands and performs specified functions; the console interpreter examines and processes all console messages and performs specified functions.

The scheduler controls the initiation and termination of all jobs on a priority basis. Disc storage allocated to the job queue is requested and released dynamically, as necessitated by the size and number of incoming jobs. When a job is terminated, all of its temporary files and other resources are released for other use. Another scheduler function is the maintenance of accounting information for each job.

The file management routines provide for the comprehensive management of files in disc storage.

The language processors translate source programs into object programs. Standard processors include CII Symbol and Meta-Symbol assemblers and the COBOL and CII FORTRAN IV compilers for 10 070 computers. Other processors are optional.

The relocating loader brings load modules (previously formed by the overlay loader) into core storage and then transfers control to the loaded program.

The overlay loader loads program elements that have been produced by a processor and are in standard object language format. It provides the program linkages needed to combine separately processed subprograms and library routines into an executable program and constructs a disc file containing the load module. If an overlay structure was specified, a table containing the structure of the program is generated.

The segment loader operates at run time to determine whether a referenced overlay segment is in core storage and, if not, to load the segment and its backward path (see "TREE", Chapter 2).

A large repertoire of Monitor functions is provided by means of user service routines. Calling sequences for such Monitor routines are included at assembly time, either explicitly or via standard Monitor procedures.

Foreground real-time programs to be included in the resident Monitor are appended to the system at System Generation time or may be created during normal batch operation. Nonresident foreground programs may be added to the system during batch operations.

System library routines are included in the system at System Generation time or may be generated and updated at run time, but they are not resident. Users' library routines may be placed in disc storage; library elements are loaded from disc storage automatically when referenced by programs or when requested as private copies in a LOAD control command.

Debug routines provide extensive capabilities for program checking and modification.

The I/O supervisor program controls the queue of outstanding I/O requests. The I/O control routines are service routines; all background tasks performing any I/O function must utilize them. Foreground tasks may also use these routines or may incorporate their own I/O routines, since they may operate in the master mode.

The cooperative and symbionts allow peripheral operations to be handled concurrently with other computing tasks, thereby maximizing CPU utilization.

DISC ORGANIZATION

Much of the Monitor system resides in disc storage when not required by current system operations, and areas of disc storage are used for system and user libraries, processors, permanent and temporary files, and scratch and checkpoint storage.

FILE ORGANIZATION

Each file is an organized collection of information that can be identified by a symbolic name. A file is either organized as a consecutive sequence of records or as a sequence of records arranged according to sort keys.

Each record of a keyed file has an identifying key associated with it. A key consists of a character string, with the first byte stating the number of characters in the string. The key associated with each record is stored at the front of that record (on tape) or in a key index (on disc). Keyed files may be accessed either directly (by key) or sequentially (by positioning the file). Records of a consecutive file can only be accessed sequentially.

SYSTEM GENERATION

System Generation is a series of processors that enables the user to generate a Monitor system tailored to the specific requirements of his installation, by processing a master system tape and a set of installation control commands.

SYSTEM HARDWARE CONFIGURATION

The minimum hardware configuration requirements for use of the Monitor are as follows:

1. 10 070 CPU with memory protection and two register blocks.
2. 24K core memory.
3. Card reader.
4. Typewriter.
5. 1.5-million byte disc unit.
6. Magnetic tape unit.

The following are strongly recommended options:

1. Card punch.
2. Line printer.
3. 32K core memory.
4. Additional Magnetic tape unit.

The resident Monitor occupies from 6,000 to about 13,000 words of core storage (depending on the hardware and software features included). The processors and libraries are retained in disc storage and loaded into core storage by the Monitor when needed. The use of disc secondary storage conserves core storage space without necessitating time-consuming manual loading operations.

SYSTEM COMMANDS

Control Command Definitions

Control Command — Definition

ASSIGN
The ASSIGN control command is used to relate an operational label or a pseudo file name to a device. A pseudo file name may be assigned to an operational label.

FMGE
Gives the user the ability to create, list, delete, copy, and punch files.

INCL
Directs the overlay loader to allocate public library routines in a segment.

JOB
Signals the completion of a previous job and the beginning of a new one. All jobs must have a JOB control command.

LIMIT
Estimates the system job parameters (i.e., number of pages of output, number of cards to be output, time job is to run, etc.) for the job.

LOAD
Directs the dynamic loader to form a relocatable load module and enters it in the user's element file if a load module name is specified.

MESSAGE
Causes the specified message to be typed to the operator when the MESSAGE command is encountered by the system.

OVERLAY
Directs the loader to form the specified overlay tree structure.

Processor Name
Tells the Monitor which processor is to operate and what options the processor is to execute.

PTREE
Tells the Monitor that a tree control command is to be read from the user's file.

RUN
Tells the Monitor to transfer control to the user's program.

STDLB
Allows the user to define system operational labels.

TITLE
Causes the specified title to be output at the beginning of each logical page of output on the LO device.

TREE
Specifies the symbolic representation of the overlay structure.

Debug Control	Definition
AND	Causes a specified test to be made at a specified location. Only if the condition is true and the specified test identifier is set does it remain set; otherwise, it is reset (see SNAPC).
COUNT	Specifies the range and the steps within the range where the test identifier is set (see SNAPC).
IF	Causes a specified test to be made at a specified location. The specified test identifier is set only if the condition is true; otherwise, the identifier is reset or remains reset (see SNAPC).
MODIFY	Allows the user to insert a modification into a foreground or background program before execution.
OR	Causes a specified test to be made at a specified location (if a specified test identifier is reset). If the condition is true, the specified test identifier is set; otherwise, it remains unchanged (see SNAPC).
PMD	Causes the Monitor to dump the selected area of memory, in hexadecimal form, if an error occurs during execution.
PMDI	Causes the Monitor to dump the selected area of memory, in hexadecimal form, regardless of whether errors have been detected.
SNAP	Causes a snapshot of the specified memory and registers at the location specified.
SNAPC	Causes a snapshot of the specified memory and registers at the location specified to be performed only when the specified test identifier is set.
SWITCH	Produces the initial settings of the pseudo sense switches.

Input Control	Definition
BCD	Serves as a terminator for a binary input source.
BIN	Informs the Monitor that the information to follow is binary.
DATA	Informs the Monitor that the information to follow is data.
EOD	Causes an end-of-data abnormal return to the Monitor, indicating the end of a series of data records.
FIN	Specifies the end of a stack of jobs.

Utility Control	Definition
PFIL	Position n files on unlabeled magnetic tape.
REW	Rewind Tape.
WEOF	Write physical end-of-file on magnetic tape.

Unsolicited Key-In Definitions

<table>
<tr><th>Key-in</th><th>Definition</th></tr>
<tr><td>ABORT|X</td><td>Terminate currently active job with no postmortem dumps.</td></tr>
<tr><td>DATE|D</td><td>Inform Monitor of current day, month, year.</td></tr>
<tr><td>DELETE</td><td>Delete specific job from system I/O file.</td></tr>
<tr><td>DISPLAY</td><td>Allows the operator to display any schedule queue as defined by the system operational label, the availability of disc, the status of the current tape units, the current operating jobs, or the time-sharing stations currently on line.</td></tr>
<tr><td>ERROR E</td><td>Terminate current job with postmortem dumps.</td></tr>
<tr><td>Fname, m</td><td>Allows foreground communication.</td></tr>
<tr><td>INT</td><td>Allows user interrupt control of his executing program.</td></tr>
<tr><td>MOUNT</td><td>Allows the operator to notify the Monitor that a required tape has been mounted.</td></tr>
<tr><td>PRIORITY</td><td>Change priority status of any job on system I/O file.</td></tr>
<tr><td>REQUEST</td><td>Allows the operator to request foreground scratch units.</td></tr>
<tr><td>SCRATCH</td><td>Allows the operator to notify the Monitor that a required scratch tape is mounted.</td></tr>
<tr><td>START|S</td><td>Causes job to continue from a wait state.</td></tr>
<tr><td>Syyndd, m</td><td>This key-in allows the operator to communicate with a symbiont.</td></tr>
<tr><td>SWITCH</td><td>Allows the operator to change the settings of the pseudo sense switches.</td></tr>
<tr><td>SYST</td><td>Allows the operator to change system parameters.</td></tr>
<tr><td>TIME|T</td><td>Informs the Monitor of current time of day.</td></tr>
<tr><td>WAIT|W</td><td>Discontinues operation on current job.</td></tr>
<tr><td>WRITELOG</td><td>Causes the Monitor to output the accounting file on the AL device.</td></tr>
<tr><td>yyndd, m</td><td>This key-in allows the operator to direct I/O recovery procedure.</td></tr>
</table>

Procedure Definitions

Procedure	Definition
M:AND	Causes a specified test to be made at a specified location. Only if the condition is true and the specified test identifier is set does it remain set; otherwise, it is reset or remains reset (see M:SNAPC).
M:ARM	Arms and connects a foreground task to a specified interrupt.
M:CAL	Connects a resident foreground task to CAL3 or CAL4.
M:CHECK	Checks type of I/O completion.
M:CHKPT	Checkpoints the job issuing the procedure on external storage.
M:CLOSE	Terminates all I/O associated with a given DCB (Data Control Block).
M:COUNT	Specifies the range and the steps within the range where a specified test identifier is set (see M:SNAPC).
M:CVOL	Causes the control program to advance to the next volume of a data set before the physical end of the current volume is detected. This call is meaningful only for tapes.
M:DCAL	Disconnects a foreground program from a CAL.
M:DCB	Defines a Data Control Block.
M:RELREC	Specifies that a data record that has been read exclusively is to be released of exclusive use.
M:DEVICE	Allows the user to set special device procedures.
M:DISABLE	Disables a specified interrupt.
M:DISARM	Disarms and connects a foreground task to a specified interrupt.
M:ENABLE	Enables a specified interrupt.
M:ERR	Returns control to the Monitor and the Monitor honors all PMD and ASSIGN control commands while ignoring all other control commands until it encounters a ! FIN or ! JOB.
M:EXIT	Returns control to the Monitor which then honors all output control commands of the form PMDI.
M:FP	Frees page of main storage owned by a given task.
M:FCP	Frees common page.
M:GL	Gets common limits.
M:GCP	Gets common pages.
M:GP	Allocates pages of main storage to the requesting task.
M:IF	Causes a specified test to be made at a specified location. Only if the specified test condition is true is the test identifier set; otherwise, it is reset or remains reset (see M:SNAPC).
M:INT	Connects a console interrupt.
M:KEYIN	Writes the specified message to the operator on the operator's console and returns the operator's reply to the program issuing the procedure.
M:LDTRC	Loads the specified load module if a reenterable copy is not available in memory, deletes the calling module, and transfers control to the loaded load module.
M:LINK	Loads the specified load module if reenterable copy is not available in memory and links to it.
M:MASTER	Causes a foreground task to change its mode of operation from "slave" to "master".
M:MERC	Allows the user to have the Monitor process any system abnormal or error code, overriding an ABN or ERR exit.
M:OPEN	Causes the specified file associated with the specified DCB to be opened.
M:OR	Causes a specified test to be made at a specified location (if a specified test identifier is reset). If the condition is true, the specified test identifier is set; otherwise, it remains unchanged (see MSNAPC).
M:PFIL	The specified tape is positioned past the number of end-of-files specified and in the direction specified.
M:PRECORD	The tape specified by the DCB will be positioned in the direction specified by the specified number of records.
M:PRINT	Writes the specified message on the listing log (LL) output media.
M:RBACK	Causes the background area checkpointed by a foreground task (via M:SBACK) to be restored.

Procedure	Definition
M:READ	Causes the next data record to be read into the location specified by the user.
M:RESTART	Restarts the specified checkpointed job.
M:DELREC	Specifies that a data record is to be deleted from the file.
M:REW	Rewinds the tape specified by the DCB.
M:RXC	Restores the user's exits to the Monitor.
M:SBACK	Allows a foreground task to checkpoint the background memory area.
M:SEGLD	Loads a specified overlay segment into memory.
M:SETDCB	Sets error or abnormal addresses in a specified Data Control Block.
M:SLAVE	Causes a foreground task to change its mode of operation from "master" to "slave".
M:SMPRT	Sets memory protection.
M:SNAP	Causes a snapshot of the registers and memory specified to be performed.
M:SNAPC	Causes a snapshot of the registers and memory specified to be performed if the specified test identifier is set. Whether the test identifier is set or not is dependent on the M:IF, M:AND, M:OR, and M:COUNT procedures.
M:STIMER	Sets the interval timer with the specified interval.
M:STRAP	Simulates a trap.
M:SXC	Supplants the user's exits to the Monitor.
M:TERM	Causes the executing foreground task to be terminated.
M:TFILE	Causes a specified Data Control Block to be closed, on return to the user's program, and the associated file to be registered as a scratch file.
M:TIME	Gives the time of day and the current date.
M:TRAP	Sets and resets the traps to go to a user routine or the standard system routine. Also sets and resets the maskable traps.
M:TRIGGER	Causes the initiation of a foreground task, by causing a specified interrupt to occur.
M:TRUNC	Causes the blocking buffer reserved for a specified Data Control Block to be released.
M:TRTN	Restores control to the user from a trap or timer routine.
M:TTIMER	Gives the time remaining in the interval that was previously set by M:STIMER and optionally cancels the interval in effect.
M:TYPE	Writes the specified message to the operator on the operator's console.
M:WEOF	Writes an end-of-file mark on the tape specified by the DCB.
M:WRITE	Causes the loaded buffer specified to be transmitted to the output device.
M:XXX	The Monitor terminates the job and does not honor any further commands until it reads another !JOB or a !FIN.

SYSTEM PARAMETERS

System parameters that may be defined during System Generation are summarized in the following table.

Disc Storage Parameters	Defaults
Number of disc tracks available	512 tracks
Number of first available disc track	Track 0
Number of disc track not to be used by system	None
Number of disc sectors per track	16 sectors
Number of disc sectors per granule	6 sectors
Number of words per disc sector	90 words
Number of tracks for symbiont queue	200 tracks
Number of tracks for permanent files	300 tracks
Number of tracks for permanent system area	12 tracks
Number of tracks for foreground files	0 tracks

Core Storage Parameters	Defaults
Size of core memory available to CPU	16K words
Number of words allocated to the Monitor's temp stack	192 words
Number of words of core storage reserved for background tasks	0 words
Number of words of core storage reserved for modification and expansion of the Monitor	0 words

Core Storage Parameters	Defaults
Number of words of core storage reserved for modification and expansion of resident foreground tasks	0 words
Number of foreground debug commands requiring resident core storage	0 commands
Number of words of core storage reserved for foreground COMMON area	0 words
Number of pages of core storage for Monitor use in file management	1 page

Buffer and Context Block Parameters	Defaults
Number of buffers pooled for Monitor use	2 buffers
Number of buffers pooled for symbiont use	1 more than the number of devices serviced by symbionts
Number of buffers pooled for symbiont context block use	1 more than the number of devices serviced by symbionts
Number of buffers pooled for current file use	4 buffers
Number of buffers pooled for foreground file indexing use	1 buffer
Number of cooperative buffers for foreground tasks	0 buffers
Number of cooperative context blocks for foreground tasks	0 blocks
Default number of buffers to be pooled for batch file indexing	1 buffer
Default.number of cooperative buffers to be allocated to batch tasks	1 buffer
External trap or interrupt is present in the Monitor system	None
Entry to specified trap or interrupt routine	None
Specified interrupt routine to be entered directly	To be entered via the Monitor's interrupt service routine
Register block n available to specified external interrupt or trap	None
Foreground task associated with a specified trap or interrupt	None
Specified foreground task is resident or nonresident	None
Relative priorities of real-time background tasks	None
Maximum number of interval timers active at one time	None
Number of tasks to be initiated by the clock interrupt	None
Register block n is available to the clock routine	Block 0

Job Limit Default Parameters	Defaults
Default limit for job execution time	5 min
Default limit for pages listed by processors for a job	100 pages
Default limit for object records produced for a job	500 records
Default limit for pages of diagnostics produced for a job	100 pages
Default limit for pages output by executing programs	100 pages
Default limit for number of granules of temporary disc storage used by a job	64 granules
Default limit for number of granules of permanent disc storage used by a job	64 granules

I/O Device Parameters	Defaults
Peripheral device name	None
Input device or output device	Both
I/O recovery tries	3 tries
I/O handlers	Standard for device
Device dedication	Undedicated
Maximum lines per page	51 lines
Maximum characters per line	132 characters
Devices associated with the symbiont	None

Miscellaneous Parameters	Defaults
Standard Monitor configuration	None
Standard operational labels	None
Resident library routines	None
Non-resident library routines	None
Master password	None
Register block n available to the Monitor	Block 0
Unimplemented instruction simulation	None
Number of I/O operations which may be queued at one time	4 I/O operations
A core map listing is to be given	No listing
Relocatable object modules are to be deleted from disc storage	Object modules are not to be deleted
Account names to be retained in resident account table	None

2. CONTROL COMMANDS

The Monitor is controlled and directed by means of control commands. These commands effect the construction and execution of programs and provide communication between a program and its environment. The environment includes the Monitor and the Meta-Symbol, Symbol, COBOL, and FORTRAN IV processors, the operator, and the peripheral equipment.

Control commands have the general form

```
! mnemonic specification
```

where

! optionally followed by one or more spaces identifies the beginning of a control message; i.e., either a control command or a control key-in (see Chapter 3).

mnemonic is the mnemonic code name of a control function or the name of a processor. It may consist of up to 8 alphanumeric characters and may begin any number of spaces after the ! character.

specification is a listing of required or optional specifications. This may include keyword operands, labels, and numeric values appropriate to the specific command. The specification field may begin one or more spaces after the mnemonic field and may consist of any number of alphanumeric characters, commas, and parentheses (see below).

In this manual, the options that may be included in the specification field of a given type of control command are identified as optional by enclosure within brackets, as shown below.

[(option 1)] [, (option 2)] . . . [, (option n)]

No brackets are actually used in control commands, and options need not appear in any particular sequence relative to each other in a specific control command. Parentheses are used to indicate the grouping of subfields, and commas are used to separate subfields, as in

[(keyword 1, value)] [,(keyword 2,value)] . . . [,(option n)]

Braces are used to identify alternative options, as shown below.

$$\left[\begin{Bmatrix}\text{(option 1)}\\ \text{(option 2)}\\ \text{(option 3)}\end{Bmatrix}\right]$$

One or more blank spaces may separate the mnemonic and specification fields, but no blanks may be embedded within a field (except in a MESSAGE control command). A period after the specification field (or after the mnemonic field if the command is one with no specification) constitutes the command terminator. If the specification field is absent and a comment follows the command, the command is terminated by a period following the mnemonic field. Annotational comments detailing the specific purpose of a command may be written following the command terminator, but no physical control command record may contain more than 80 characters, including blanks.

The specification field may be continued from one physical record to the next by the use of a semicolon, as indicated by the following example:

```
! fication comment

! mnemonic speci;
```

Although a comment field may also be continued from one record to the next, no comment in a control command record may contain a semicolon (except as a continuation indicator).

Communication between the operator and the Monitor is accomplished through control commands, key-ins, and messages. Control commands are usually input to the Monitor via punched cards, and control key-ins are always through the operator's console; however, any input device(s) may be designated for these functions (see "ASSIGN", below). All control commands and Monitor messages are listed on the output device designated as the listing log (normally a line printer). In this manner, the Monitor keeps the operator informed about the progress of each job.

Control commands may be categorized as follows:

System		Debug
JOB	COBOL	PMD
LIMIT	LOAD	PMDI
POOL	OVERLAY	SNAP
STDLB	INCL	SNAPC
MESSAGE	TREE	IF
TITLE	PTREE	AND
ASSIGN	MODIFY	OR
FMGE	SWITCH	COUNT
SYMBOL	RUN	
METASYM		
FORTRAN		

Input	Utility
BIN	PFIL
BCD	REW
EOD	WEOF
DATA	
FIN	

SYSTEM CONTROL COMMANDS

JOB Each background job to be processed by the system must begin with a JOB control command. The JOB command signals the completion of the previous job, if any, and the beginning of a new one. Standard system assignments become effective when a JOB control command is encountered.

The form of the JOB control command is

```
! JOB account number, name [, priority]
```

where

account number identifies the account or project. It consists of from 1 to 8 numeric characters.

name identifies the user. It consists of from 1 to 12 alphanumeric characters.

priority specifies the priority of the job (0-F_{16}), where F is the highest (i.e., most urgent) priority. If no priority is specified, the default value is 1.

Example:

```
! JOB 12345, JOBSAMP1, 1
```

The above example specifies that the account number for the job is 12345, the user is JOBSAMP1, and the job has priority 1. The JOB control command may not be continued from card to card. Fields must be terminated by commas.

LIMIT If a LIMIT control command is included in a job, it must follow the JOB control command immediately. It is used to specify maximum values for various system resources used by the job. If some parameters are not specified in a LIMIT control command, or if the LIMIT command is omitted from the job (or does not follow the JOB command immediately), the limits established during System Generation will apply for the unspecified parameters. If any limit is exceeded when the job is run, the job is aborted and the Monitor skips to the next JOB control command.

The form of the LIMIT control command is

```
! LIMIT (option)[, (option)]...[, (option)]
```

where the options are

TIME,value[†] specifies, in minutes, the maximum execution time for the current job.

LO,value specifies the maximum number of pages (excluding diagnostic output) that may be listed for programs processed in the current job (processor output plus system output).

PO,value specifies the maximum number of object records that may be produced in the current job.

DO,value specifies the maximum number of pages that may be output for diagnostics in the current job.

UO,value specifies the maximum number of pages that may be output (on the LO device) by executing program(s) in the current job (user's executing program).

TSTORE,value specifies, in granules, the maximum amount of temporary direct-access storage (on disc) that may be used by the current job.

PSTORE,value specifies, in granules, the maximum amount of permanent direct-access storage (on disc) that may be used by the current job.

SCRATCH,value specifies the maximum number of scratch tapes that may be used at any one time by the current job.

Example:

```
! ,(SCRATCH,2)
! (TSTORE,10),(PSTORE,20);
! (PO,2500),(DO,50),(UO,75),;
! LIMIT (TIME,10),(LO,100),;
```

The above example specifies that the current job may require no more than: 10 minutes of execution time, 100 pages of object listings, 2500 object cards, 50 pages of diagnostic output, 75 pages of output produced by the executing program, 10 granules of temporary disc storage, and 20 granules of permanent disc storage. It also specifies that not more than 2 scratch tapes are to be used at any one time.

POOL A POOL control command may appear anywhere in a job except between the JOB command and the LIMIT command or imbedded in a series of debug commands. It is used to specify the number of buffers to be allocated for file management and file indexing used by the Monitor. If not specified, system limits are assumed. The maximum number of buffers allocated will never exceed available memory.

[†] ALL values in a LIMIT control command are expressed as decimal integers.

The minimum will always be 1 (for each pool). The buffer area will be allocated from the background area after each processor or user's program is loaded for execution.

The form of the POOL control command is

```
! POOL    (option)[,(option)]
```

where the options are

FPOOL,value   specifies the number of 512-word buffers to be assigned to file management use. These buffers are used by the Monitor for packing and unpacking data. Each active file must use a data buffer as well as an index buffer (see IPOOL below). If there are fewer buffers in the pool than there are active files, the buffers will be shared.

IPOOL,value   specifies the number of buffers to be assigned to the file index pool. Each IPOOL buffer is twelve words larger than the sector size of the disc. For a 90-word/sector disc, 5 index buffers will fit into one page.

STDLB A STDLB control command may be used to define or redefine (for the duration of the current job) any specified system operational label, with the exception of OC (operator's console). An operational label is a symbolic name used to identify a logical system device. Standard Monitor operational labels are given in Table 1. STDLB control commands may also be used to redefine foreground operational labels that were defined initially during System Generation.

Table 1. Monitor Operational Labels

Label	Reference	Comments
BI	Binary input	Binary coded input will be received from the device to which this label is assigned.
CI	Compressed input	Compressed symbolic input will be received from the device to which this label is assigned.
EI	Element input	Element file input will be received from the device to which this label is assigned.
SI	Source input	Symbolic (source language) input will be received from the device to which this label is assigned.
C	Control input	Input from the device to which this label is assigned will be monitored, so that all control commands will be recognized by the Monitor.
BO	Binary output	Binary coded output will be transmitted to the device to which this label is assigned.
CO	Compressed output	Compressed symbolic output will be transmitted to the device to which this label is assigned.
DO	Diagnostic output	Diagnostic program dumps will be output on the device to which this label is assigned.
EO	Element output	Element file output will be transmitted to the device to which this label is assigned.
LO	Listing output	Source and object listings for assemblies and compilations will be output on the device to which this label is assigned.
SO	Source output	Symbolic (source language) output will be transmitted to the device to which this label is assigned.
PO	Punch output	BCD or binary coded output will be transmitted to the device to which this label is assigned (normally a card or paper tape punch).
AL	Accounting log	The accounting file will be output on the device to which this label is assigned, as a response to a WRITELOG key-in.
LL	Listing log	All control commands and system messages, including accounting information for the job, will be output on the device to which this label is assigned.
OC	Operator's console	All JOB, MESSAGE, and FIN control commands, and all job termination messages will be output on the device to which this label is assigned (OC may not be assigned to another operational label).

The form of the STDLB control command is

```
! STDLB operational label, device name
```

where

operational label   specifies either a Monitor operational label (other than OC) or a foreground operational label. An operational label consists of one or two alphanumeric characters.

device name   specifies either an operational label or a physical device name of the form yyndd

where

yy   specifies the type of device (see Table 2).

n   specifies the channel letter (see Table 3).

dd   specifies the device number (see Table 4), in hexadecimal.

Table 2. I/O Device Type Codes

yy	Device Type
MT	Magnetic tape
7T	7-track magnetic tape
9T	9-track magnetic tape
CP	Card punch
CR	Card reader
PP	Paper tape punch
PR	Paper tape reader
TY	Typewriter
LP	Line printer
MD	Magnetic drum
DC	Magnetic disc
PL	Plotter
NO	No device

Table 3. Channel Designation Codes

Specified Channel Letter (n)	Corresponding Decimal Digit of Unit Address
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7

Table 4. Device Designation Codes

Hexadecimal Code (dd)	Device Designation
00 ≤ dd ≤ 7F	Refers to a device number (00 through 7F).
80 ≤ dd ≤ FF	Refers to a device controller number (8 through F) followed by a device number (0 through F).

Examples:

```
! STDLB  LO,LPA15
```

The above example specifies that the operational label LO is to be assigned to a line printer: device number 15 on channel A.

The following example specifies that the operational label DO is to be assigned to the device to which the operational label LO is assigned.

```
! STDLB  DO,LO
```

MESSAGE   A MESSAGE control command may be used to type a message to the operator at the time that it is encountered by the Monitor. The form of the MESSAGE control command is

```
! MESSAGE  message string
```

where

message string   specifies the message to be typed. The message string may contain any desired characters, including blanks, but may not be continued from one record to the next. Two or more MESSAGE control commands may be used in immediate succession.

Example:

```
! MESSAGE SEND ALL SAVE TAPES TO TOM ATKINS
```

The above example would cause the following message to be output on the LL and OC devices.

```
!! MESSAGE SEND ALL SAVE TAPES TO TOM ATKINS
```

Note: All Monitor messages to the operator begin with two exclamation characters.

TITLE A TITLE control command may be used to produce a heading at the beginning of each logical page listed on the LO device. The form of the TITLE control command is

```
! TITLE title string
```

where

title string specifies the title that is to appear on each page. The title string may contain any desired characters, including blanks, but may not be continued from one record to the next.

Example:

```
! TITLE CRITICAL PATH ANALYSIS RP-18
```

The above example would cause the title string to be output with every logical page listed on the LO medium by the executing program. The entire control command would also be output (once) on the LL device, as are all control commands.

If more than one TITLE control command is used in a job, the one most recently encountered by the Monitor supersedes, and page numbering begins with a 1 whenever a TITLE control command occurs.

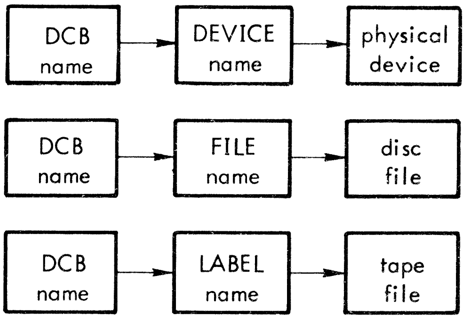
ASSIGN ASSIGN control commands specify what files and physical peripheral devices are to be used in the current job, and the uses to which they will be put. ASSIGN commands appear prior to a LOAD or OVERLAY control command. They may also occur elsewhere within a job, if new assignments are to be made after part of the job has been processed. Each ASSIGN command assigns a Data Control Block (DCB) name to a file name[†] or device name. An operational label is a symbolic name used to identify a logical system device (see Table 1). A pseudo file name is a symbolic name used (in a user's program) to identify a logical device or file.

The "device name" to which a DCB name may be assigned may be either a physical device name (see "STDLB") or a logical device name (i.e., an operational label). For example, a DCB name corresponding to a pseudo file name may be assigned to an operational label which in turn may have been assigned (via a STDLB control command) to some physical device name or to another operational label.

The "file name" to which a DCB name may be assigned may correspond to a disc file or a magnetic tape file. The type of file is specified by the keyword used with the file name.

Output to a labeled tape or disc file through a Monitor DCB such as M:BO, M:LO, etc., will exist as a single file provided that the DCB is <u>not</u> reassigned between job steps via an ASSIGN control command or an M:OPEN procedure call (see "M:OPEN", Chapter 5).

The three general types of assignments that may be made via an ASSIGN command are illustrated below.



The form of the ASSIGN control command is

```
! ASSIGN dcb name,(option)[,(option),...,(option)]
```

where

dcb name specifies the name (not exceeding 31 characters in length) by which the DCB may be referenced. This must be the first subfield following ASSIGN, and must be followed by at least one of the optional specifications given below. The first two characters of a user's DCB name must be "F" and ":" (e.g., F:PRINT or F:BI). The first two characters of a Monitor or foreground DCB name are "M" and ":" (e.g., M:LO).

The options are as follows:

<u>name</u> (one of the three keyword operands given below).

1. DEVICE,name specifies a system physical device name or a system operational label. Specific magnetic tape units should not be assigned by their physical device name. However, magnetic tape may be designated by one of the three specifications listed below.

   a. DEVICE,MT specifies that the Monitor is to assign the specified DCB to any available magnetic tape unit.

   b. DEVICE,7T specifies that the Monitor is to assign the specified DCB to any available 7-track magnetic tape unit.

   c. DEVICE,9T specifies that the Monitor is to assign the specified DCB to any available 9-track magnetic tape unit.

   If no serial number is specified (see INSN and OUTSN, below) and the DCB is opened in the output mode, an available scratch tape of the type specified will be used. If no serial number is

[†] Up to 31 alphanumeric characters.

specified and the DCB is to be opened in the input mode, the DCB is not opened and an abnormal return is given, indicating insufficient parameters.

2. FILE,name[,account] specifies the name of the file (i.e., the system file directory name). The named file will be maintained in disc storage. If the named file belongs to a different account than that of the current job, the file's account number must be given.

3. LABEL,name [,account] specifies the name of the file (i.e., the name identifying the file on magnetic tape). An INSN or OUTSN option (see below) must be used to specify the particular tape(s) containing the file. If the named file belongs to a different account than that of the current job, the file's account number must be given.

org (one of the two file organization types given below).

1. CONSEC specifies that the records in the file are consecutively organized and each record is to be processed in order.

2. KEYED specifies that the location of each record in the file is determined by an explicit identifier (key) that may be used to address the device containing the file.

access (one of the two record access means given below).

1. SEQUEN specifies that records in the file are to be accessed in the order in which they appear within the file.

2. DIRECT specifies that the next record to be accessed is to be determined by an explicit identifier (a "key", see Chapter 5).

function (one of the four file modes given below).

1. IN specifies the file input mode.

2. OUT specifies the file output mode.

3. INOUT specifies the file input and output mode (i.e., the update mode).

4. OUTIN specifies the file output and input mode (i.e., the scratch mode).

PASS,value specifies the password that will allow access to a classified data file (after any other security checks have been made). The password may be from 1 through 8 alphanumeric characters in length and will be omitted from the listing of the ASSIGN command.

file (one of the two specifications given below).

1. REL specifies that, for single-file tapes, the reel is to be released to the Monitor's scratch files. For multi-filed tapes, no action is to occur if the tape is being used for input; on output, the tape is to be positioned at the beginning of the file (allowing the file to be overwritten). For direct access devices (i.e., disc), all allocated direct access storage for the file is to be released to the Monitor.

2. SAVE specifies that the file is to be included in the system file directory. If the file function (see above) is OUT or OUTIN, the SAVE option must be specified (either in the ASSIGN control command or in an M:DCB or M:OPEN procedure call; see Chapter 5, to cause the Monitor to allocate permanent disc storage for the file. When closing such a file, SAVE must again be specified, in the M:CLOSE procedure call (see Chapter 5), if the file is to be permanently saved in disc storage.

READ,value[,value]... [,value] specifies the account numbers of those accounts that may read but not write the file. The value "ALL" may be used to specify that any account may read but not write the file (e.g., READ, ALL). The value "NONE" may be used to specify that no account may read the file. If no value is specified, or if READ (and WRITE, see below) is omitted, ALL is assumed by default. The total number of accounts explicitly specified in a READ or WRITE specification must not exceed 8.

WRITE,value[,value]... [,value] specifies the account numbers of those accounts that may have both read and write access to the file. The values "ALL" and "NONE" may be used, as with the READ option (see above); and, if a conflict exists between READ and WRITE specifications, those of the WRITE option take precedence. NONE is assumed by default.

INSN,value [,value] [,value] specifies the serial numbers of magnetic tapes that are to be used for file input. · These numbers must be ordered in the proper sequence for the file. A maximum of three values may be specified for Monitor DCBs; for the user's DCBs, the number of values is not limited. Serial numbers may be from 1 to 4 alphanumeric characters in length. INSN has no effect on file mode.

OUTSN,value[,value] [,value] specifies the serial numbers of magnetic tapes that are to be used for file output. If the output fills the first reel, then the second reel specified will be used, etc. A maximum of three values may be specified for Monitor DCBs; for the user's DCBs, the number of values is not limited. Serial numbers may be from 1 to 4 alphanumeric characters in length. OUTSN has no effect on file mode.

RECL,value specifies the maximum record length, in bytes. The greatest value that may be specified is 32,767. If RECL is not specified, a standard value (appropriate to the type of device used) will apply by default.

TRIES,value specifies the maximum number of recovery tries to be performed for any I/O operation. The greatest value that may be specified is 255.

KEYM,value specifies the maximum length, in bytes, of the keys associated with records within the file. If KEYM is not specified, the value 11 is assumed.

VOL,value specifies the beginning volume number of a multivolume file.

The following options are device-dependent, and will be ignored by the Monitor in all cases where they are not applicable to the device used.

format (one of the two following specifications).

1. VFC specifies that the first character of each record is a format-control character for printing (see Table 5).

Table 5. Line Printer Format Control Codes

Code (hexadecimal)	Action
C0,40	Space no additional lines.
60,E0	Inhibit space after printing.
C1	Space 1 additional line before printing.
C2	Space 2 additional lines before printing.
C3	Space 3 additional lines before printing.
⋮	⋮
CF	Space 15 additional lines before printing.
F0	Skip to Channel 0 (bottom of page) before printing.
F1	Skip to Channel 1 (top of page) before printing.
F2	Skip to Channel 2 before printing.
⋮	⋮
FF	Skip to Channel 15 before printing.

2. NOVFC specifies that the records do not contain format-control characters.

COUNT,tab specifies that a page count is to appear at the top of each page, beginning in the column specified by "tab". If COUNT is specified for the LO device and TITLE is also specified, the page count will be superimposed on the title line.

Example:

COUNT,60

The above example specifies that the most significant digit of the page count is to appear in column 60 at the top of each page.

DATA,tab specifies that output is to begin (on each page) in the column specified by "tab". Data exceeding device line length will be lost.

SEQ[,id] specifies that the punched output is to have sequencing in columns 77-80. If id is specified, it will appear in EBCDIC, in columns 73-76 of the punched output. Sequencing begins with 0000 and is incremented by 1 for each record.

LINES,value specifies the number of printable lines per page. The greatest value that may be specified is 127. If LINES is not specified, the value established at System Generation time will apply.

SPACE,value[,top] specifies the spacing between lines (value) and between the top of each page and the first line printed (top). A value of 1 indicates that lines are to be single-spaced. The greatest value that may be specified is 15.

mode (any of the following specifications for I/O mode).

1. BCD specifies that EBCDIC mode is to be used.
2. BIN specifies that the binary device mode is to be used.
3. FBCD specifies that FORTRAN BCD conversion is to be used.

   If no mode is specified, the current mode established for the DCB (see Chapter 5) is used.
4. PACK specifies that the packed binary mode (7-track tape) is to be used.
5. UNPACK specifies that the unpacked binary mode (7-track tape) is to be used.
6. L specifies that a listing-type device is to be used.

BIN/BCD controls the mode of writing to CP,PP, or 7T, and reading from 7T. It also controls the mode of reading from CR if DIRECT has been specified.

FBCD causes conversion from the FORTRAN BCD set to EBCDIC on reading CR or 7T and the opposite conversion when writing to CP or 7T.

PACK/UNPACK specifies packed or unpacked binary on 7T if BIN is also specified.

If no ASSIGN commands are included in a job, standard Monitor input/output assignments will apply during that job. Standard I/O assignments are specified in the Monitor system tape (or deck) and, when the Monitor system is loaded into resident storage, these assignments are established and remain in effect until altered by an ASSIGN command or a SYST key-in (see Chapter 3). Standard assignments altered by ASSIGN commands revert to standard at the end of a job, but assignments altered by SYST key-ins replace standard assignments until the system is reinitialized by loading it again from the Monitor system tape (or until changed by another SYST key-in).

Examples:

1. A labeled multi-reel tape file:

```
! ASSIGN  F:TAPE,(LABEL,ABC),(INSN,256,231,001)
```

This example specifies that the user's DCB name F:TAPE is to be assigned to magnetic tape file name ABC. It also specifies that input tapes 256, 231, and 001 are to be used (in that order) to input the file.

2. The system listing output medium:

```
! ASSIGN  F:OUT,(DEVICE,LO),(SEQ,OUT)
```

This example specifies that the user's DCB name F:OUT is to be assigned to the output device to which the operational label LO is assigned (normally a line printer). Sequence numbers are to be printed in columns 77-80 and the identification "OUT" is to be printed in columns 73-76 of each record.

3. A listed output file:

```
! ASSIGN  F:JED,(FILE,XX1A),(EXPIRE,NEVER)
```

This example specifies that the user's DCB name F:JED is to be assigned to the disc file name XX1A, and also specifies that the file is to be retained in disc storage indefinitely.

FMGE A file management (FMGE) control command may be used to enter, copy, list, delete, or punch any file in the system file directory. The M:EO and M:EI DCBs are used to enter information and obtain information, respectively. The entire user's file directory will be listed if no options are specified. The entire file directory, as defined by the current M:EI assignment, will be listed; that is, any account's file directory may be listed by assigning the M:EI DCB to that account.

The form of the FMGE control command is

```
! FMGE  [(option)] ... [,(option)]
```

where the options are

DELETE specifies that the file is to be deleted from the system file directory after any other action specified by the FMGE command has been taken. The file that is to be deleted is determined by the current assignment of the M:EI DCB.

LIST[,mode] specifies that the file is to be listed on the LO device. The mode may be either FORMAT, BIN (binary), or BCE (EBCDIC). The file that is to be listed is determined by the current assignment of the M:EI DCB. If BIN is specified, a hexadecimal dump of each record is given on the DO device. If FORMAT is specified, BCD is assumed and the first byte of each record is assumed to be a VFC byte. The default option is BCD. If the file is keyed, the keys are also listed.

PUNCH[,mode] specifies that the file is to be output on the PO device. The mode may be either BIN (binary) or BCD (EBCDIC). The default option is BCD. Any element file punched may be reintroduced to the system by means of the ENTER option (see below). The file punched is determined by the current assignment of the M:EI DCB.

ENTER[,PERM] specifies that an element obtained from the EI device is to be added to the system file directory. If PERM is also specified, the file is placed in permanent disc storage; otherwise, the element is deleted from disc storage when the job is terminated. The information read from the EI device is written on the EO device or file.

Example:

```
! FMGE  (DELETE),(PUNCH,BIN)
```

This example specifies that the file assigned to the M:EI DCB is to be deleted from the system file directory after it has been output on the PO device in the binary mode.

PROCESSOR CONTROL COMMANDS

A processor control command indicates to the Monitor that control is to be transferred to the specified resident processor. It also specifies the types of input to be accepted, and the types of output to be produced by the processor.

Processors may be created, updated, and deleted under normal batch operations. There are no restrictions as to how many and what kind of processors may be added to the system.

User programs are called by ! RUN(LMN,name) and processors are called by ! name, where the name is the name of the load module specified in the LOAD or OVERLAY control command. User's processors must be inserted under the Monitor system account number.

A LOAD or OVERLAY control command normally follows a processor command (and is read after all specified inputs have been received and processed), so that the processor's output will be translated into an executable load module.

The form of a processor control command is

```
! name  specification
```

where

name is the name of a Monitor processor (e.g., FORTRAN, SYMBOL, COBOL or METASYM).

specification is a listing of input and output options for the processor. At least one input option and one output option must be specified. The following lists the options that may be specified.

Specification	Specification Reference
CI	Compressed input from the CI device
SI	Symbolic input from the SI device
BO	Relocatable binary output (on cards or paper tape) on the BO device
CO	Compressed output on the CO device
GO	Relocatable binary output to disc storage (i.e., the GO file)
LO	Listing output on the LO device
LS	Listing source
D	Debug output
S	S in column 1 (FORTRAN only)

Example:

```
! METASYM SI,LO,GO
```

This example specifies that control is to be given to the Meta-Symbol assembler. It also specifies that symbolic input will be received from the device to which the SI operational label is assigned, listing output is to be transmitted to the device to which the LO operational label is assigned, and relocatable binary output is to be stored in the file to which the M:GO DCB is assigned.

LOAD A LOAD control command is used to direct the loader to form a relocatable load module (i.e., an executable program) from relocatable object modules (i.e., subprograms in Sigma object language) and library load modules. The loader will also generate DCBs to be included in the load module (see Appendix T). The object modules or load modules may be loaded from one or more of the following:

1. The GO file(s).
2. Element files.
3. The system library.
4. The BI medium.

The resulting program may be entered into the user's element file, and thereafter called internally by an executing program, or it may be executed independently, as specified by a RUN control command.

The form of the LOAD control command is

```
! LOAD [(option)] [,(option)]...[,(option)]
```

where the options are

GO specifies that data from the user's temporary GO file is to be included in the root of the load module (see "TREE" below).

EF,(name[,account [,password]])[,...] specifies that the named module (either object or load) from the element file of the designated account is to be included in the load module. If no account number is specified, that of the current job is assumed. If a password is associated with a named module, it and the account number must be included in the specification. More than one module may be specified in an EF specification.

UNSAT,(account[,password])[,...] specifies that the library of the designated account is to be searched for external definitions required for the load module (i.e., corresponding to primary external references). More than one account may be specified in an UNSAT specification. The library password (if any) for each account must be included in the specification, although listing of the password is suppressed. (Library passwords are defined by PERM specifications, see below.)

PERM[,LIB] specifies that the load module is to be added to the account's element file. If PERM is omitted, the load module will be placed in the account's temporary file. If a previously formed load module of the same name (see LMN, below) exists, it will be replaced by the newly formed one. If LIB is specified, any external definitions or external references in the load module will be added to the account library's table of external definitions and references, and the load module will be inserted into the library.

LMN,name [,password] specifies the name that is to be given to the load module. If no name is specified for a load module, it is considered temporary, even if PERM (see above) is specified. A password to be associated with the load module may be specified.

If (PERM,LIB) is specified, the password is the password of the library. The password specified for the first library load module entered in the library becomes the password of the library.

READ
WRITE } (see ASSIGN control command).
EXPIRE

SL,value specifies the error severity level that will be tolerated by the loader in forming a load module. The value may have the range shown in Table 6.

TSS,size specifies (in hexadecimal) the maximum size, in words, of the temporary storage stack for the current job (see Chapter 7). If TSS is omitted, the maximum size is set at 10 words. The greatest size that may be specified is limited to available core storage and may not exceed 7FFF words regardless of core size.

BIAS,value specifies (in hexadecimal) the load bias, in word locations. If the value is not a page boundary, the next lower page boundary is used.

Table 6. Error Severity Levels

Error Severity Level	Typical Meaning (Actual Meaning is Determined by the Processor Used)
0	No abnormalities detected.
1	Abnormality detected but error unlikely.
2	Possible error detected.
3	
4	Probable error detected.
5	
6	
7	Definite error detected, but localized in effect: e.g., an undefined symbol.
8	
9	

Table 6. Error Severity Levels (cont.)

Error Severity Level	Typical Meaning (Actual Meaning is Determined by the Processor Used)
A	
B	Definite error detected, not localized in effect; e.g., improper DO loop nesting.
C	
D	
E	Disastrous error.
F	

NOSYSLIB specifies that the system library is not to be searched. If NOSYSLIB is omitted, the system library will be searched to satisfy any external references that are unsatisfied after loading has been accomplished from all other specified sources.

MAP If MAP is specified, a complete listing of external references and definitions for the load module is to be output on the LL device (see Appendix L).

BI specifies that the BI input device is to be used to read unspecified relocatable object modules. Object modules will be loaded from the BI device until either two end-of-data codes (05) or one end-of-file code (06) is encountered. If neither BI, EF, nor GO are specified as input sources, BI is assumed by default. Normally, the BI and C operational labels are both assigned to the same device. If a control command is read, the Monitor generates an end-of-file code and terminates the binary input.

M10 specifies that each control or dummy section is to be loaded at the next greater multiple of 10_{16}.

M100 (same as M10, above, except that loading starts at the next greater multiple of 100_{16}).

ABS specifies that a relocation dictionary is not to be formed for the load module. If ABS is omitted, a relocation dictionary will be formed and the load module will be treated as "semiabsolute" (i.e., executable but capable of being relocated).

ERTABLE,size specifies the size, in words, of the library error table (see the Mathematical Routines user's manual, publication C 900906. The default option is 10 words.

ERSTACK,size specifies the size, in words, of the library error stack. The default option is 10 words.

NOTCB specifies that no Task Control Block (TCB) is to be created by the loader. This option should not be used for FORTRAN jobs, since FORTRAN requires a TCB.

Any number of LOAD commands may be used in a single job.

Examples:

```
! LOAD
```

This example specifies that loading is to be accomplished from the BI device. Default conditions are assumed for all options.

```
! (BI), (M100), (FCOM)
! (SL, 2), (TSS, 3E8), (BIAS, 2000), ;
! (WRITE, NONE), (EXPIRE, NEVER), ;
! (LMN, ANI), (PERM, LIB), (READ,ALL), ;
! (UNSAT, (1235), ;
! LOAD  (EF, (ABC, 1234, AMT)), ;
```

This example specifies that:

1. No load information is to be taken from the GO file, since GO is not specified.
2. Element ABC, having the password AMT associated with it, is to be loaded from the element file of account 1234.
3. The element file of account 1235 is to be searched for external definitions corresponding to unsatisfied external references (if any exist after loading has been accomplished from all other specified sources).
4. The name ANI is to be associated with the load module being formed.
5. The load module is to be a permanent file in the user's library.
6. Any account may read the load module, but none may write into it.
7. The disc storage allocated to the load module is never to be released for other use.
8. Errors of severity level 2 are acceptable in the load module.
9. Up to $3E8_{16}$ words of temporary storage may be used by the current job.
10. A relocation bias of 2000_{16} is to be used.
11. No load map is to be output.
12. Relocatable object modules are to be loaded from the BI input device.
13. Each control section or dummy section is to be loaded starting at a multiple of 100_{16}.

OVERLAY CONTROL COMMANDS

The use of an overlay program structure allows two or more program segments to begin at the same logical core storage location at different times during program execution. This capability is often useful when the total storage requirements of a program exceed the size of available core storage. The formation of the load module for an overlay program is accomplished in the same way that load modules in general are formed (see "LOAD", above). However, the load parameters must be specified either in an OVERLAY control command (see below) or a LOAD command, and the overlay structure must then be specified in a TREE control command (see below). There is no functional difference between the LOAD and OVERLAY commands. The SEG and REF options may be used in a LOAD command and a TREE command may follow a LOAD command also. Cross referencing between paths (see "TREE" below) is not allowed.

Debug control commands may be used with overlay programs.

OVERLAY An OVERLAY control command may be used to specify the load parameters for an overlay load module. The load parameters that may be specified in OVERLAY commands include those appropriate to LOAD control commands. Also, either of two overlay modes (see SEG and REF, below) may be specified.

The form of the OVERLAY control command is

```
! OVERLAY [(option)] [,(option)] . . . [,(option)]
```

where the options include any of those appropriate to the LOAD control command and either of the following:

SEG specifies that the overlay structure is to be set up for the segment loading mode. In this mode, it is the user's responsibility to explicitly load each segment from disc storage to core storage (e.g., by means of the M:SEGLD procedure; see Chapter 4) before it is referenced by the executing program.

REF specifies that the overlay structure is to be set up for the reference loading mode. In this mode, any permissible reference (in another segment of the program) to an external definition within a given segment will cause that segment and all its backward path (see "TREE" below) to be loaded, if it is not already in core storage, even in the case of an unsatisfied conditional branch to that segment. The external reference must not originate in an alterable instruction (i.e., one that may be replaced or changed during program execution).

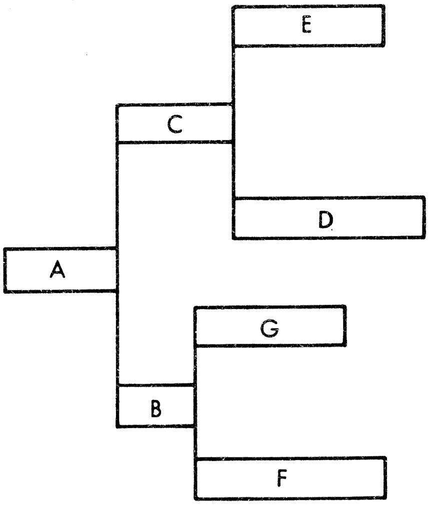
If neither REF nor SEG is specified, segment loading (see SEG, above) is assumed.

TREE A TREE control command must appear immediately following the associated OVERLAY command. It must specify the overlay structure of the load module formed as a result of the preceding OVERLAY command, so that the logical segments of the program will be loaded from disc storage into core storage as required. It is the user's responsibility to plan the relationship of these segments.

The relationship of the segments that comprise an overlay program can be represented graphically by means of a tree diagram, as in the example shown below. The horizontal coordinate of the diagram denotes increasing core storage (address) allocation, from left to right. The vertical coordinate denotes overlays. The leftmost segment, or "root", is that portion of the program that resides in core storage through program execution. A "path" of an overlay consists of those segments that may occupy core storage at the same time. The portion of a path that extends from the start of the program (i.e., the root) to a given segment is termed the "backward path" of that segment.

The following example consists of four paths, any one of which may be present in core storage at any given time. Segment A, below, is the root of the program and is never overlaid by another segment. Any path may be loaded into core storage and overlaid as many times as required by the program. All segments of the load module are saved in disc storage and, when a segment that has been overlaid is called again by the executing program, the original copy is loaded from the disc. Therefore, any communication between two overlay segments (e.g., D and E, below) must be done in a part of the backward path common to both.

Example:



The form of the TREE control command is:

```
! TREE specification
```

where

specification specifies the tree structure by use of the symbology given below.

name specifies the name of a relocatable object module.

- indicates that two named relocatable object modules are to be contiguous in core storage.

, indicates that two segments are to overlay one another (i.e., begin at the same core storage location.

() indicates a new (lower) level of overlay.

Example:

```
! TREE  A - (C - (E,D), B - (G,F))
```

The above example is a symbolic representation of the overlay structure of the preceding graphic example.

PTREE A PTREE control command may be used to obtain a TREE control command from the user's file.

The form of the PTREE control command is

```
! PTREE  (name[,account [,password]] )
```

where

name specifies the name of the file containing the TREE control command. This file can be created by ENTERing (see "FMGE") a TREE control command without an exclamation character.

account specifies the account containing the designated file.

password specifies the password associated with the designated file. If the file has an associated password, both it and the account number must be given in the command.

INCL An INCL (include) control command may be used, following a TREE or PTREE command, to include a named public library routine in a specified overlay segment (e.g., to satisfy a secondary external reference).

The form of the INCL control command is

```
! INCL,segment  name [...,name]
```

where

segment specifies the name of the segment to which the named library routine is to be appended. Each segment takes the name of the first element file named in the segment specified on the TREE card.

name specifies the name of a public library routine that is to be appended to the specified segment.

Primary external references to public library routines are satisfied automatically by the overlay loader. Therefore, it is not necessary to specify such references in an INCL command. However, there is no restriction against doing so. Any number of public library routines may be specified in a single INCL command.

OVERLAY EXAMPLE

An example of the control card sequence used to specify the structure of an overlay program is given below.

```
! INCL, AIRT  L:LOG
! INCL, BOSK  L:SIN, L:COS
! TREE AIRT - (BOSK, KALX - ABC)
! (EXPIRE,7,11,73), (BI), (REF), (MAP)
! LIB), (READ, 1234), (WRITE, NONE),;
! (UNSAT, (1236), (1237)), (LMN, OMER),(PERM,;
! OVERLAY  (EF, (ABC), (AIRT), (BOSK), (KALX)),;
```

The above example specifies that:

1. No load information is to be taken from any GO file.
2. Elements ABC, AIRT, BOSK, and KALX are to be loaded from the element file of the present job.
3. The element files of accounts 1236 and 1237 are to be searched if unsatisfied external references exist after loading has been accomplishe for all other sources specified.
4. The name OMER is to be associated with the load module being formed.
5. The load module is to be a permanent file in the user's library.
6. Account 1234 may read the load module, but no account (other than that of the current job) may write into it.
7. The disc storage allocated to the load module is to be released for other use on July 11, 1973
8. Relocatable object modules are to be loaded from the BI input device and are to be included in the root segment (AIRT).
9. The overlay structure is to be set up for loading in the reference mode.
10. A load map is to be output.
11. The system library is to be searched for external definitions corresponding to unsatisfied primary external references (if any).

OLAY The OLAY control command provides the user with a larger background area in which to load than do the LOAD and OVERLAY control commands. The OLAY command uses an overlaid version of the loader, whereas both LOAD and OVERLAY use an unsegmented version. The unsegmented version is faster, but the overlaid version takes less core space.

The OLAY control command may be substituted for either the LOAD or OVERLAY control command. All options used with LOAD or OVERLAY also may be used with OLAY and the control command sequence may be identical to that used with a LOAD or OVERLAY command.

MODIFY The MODIFY control command allows the user to insert words into a foreground or background program in core storage.

The form of the MODIFY control command is

```
! MODIFY[,segment] loc,word[,word]... [,word]
```

where

segment specifies the name of an overlay segment. This parameter is omitted if the load module is not overlaid.

loc specifies a relative hexadecimal location (i.e., an external definition followed by an optional hexadecimal addend value) or a signed positive absolute hexadecimal address where the modification is to be made. If an external definition is used, and the modification is to be made to an overlay segment, the definition must not have been referenced in a "lower" segment of the overlay tree. This restriction applies only if the MODIFY command appears after the OVERLAY command for the program.

word specifies the word to be inserted (right-justified) at the designated location (see "loc", above). The word must be expressed as an unsigned hexadecimal (i.e., value+name). If it is desired to specify an address resolution for the external definition (following the value), the name of the external definition must be enclosed in parentheses (i.e., value+res (name)).

res	Resolution
BA	Byte
HA	Halfword
WA	Word
DA	Doubleword

If no resolution is specified, word resolution is assumed.

The MODIFY control command may be used either following a LOAD command or a RUN command (see "RUN", below). If used following a LOAD command, the inserted words become a permanent part of the program; otherwise, they are a temporary "patch" used only during the current execution of the program.

Example:

```
! MODIFY LOC1+A1,1234E
```

This example specifies that the hexadecimal value 1234E is to be inserted at a location whose address is 161 words higher than that of LOC1.

SWITCH Any of six pseudo sense switches (see Appendix D) may be set or reset by means of the SWITCH control command (or by an unsolicited key-in; see Chapter 3).

The form of the SWITCH control command is

```
! [,(RESET,value[,...,value])]
! SWITCH [(SET,value[,...,value])];
```

where

SET,value specifies which pseudo sense switches are to be set. The "value" may be from 1 to 6, and more than one value may be specified. If ALL is specified, all pseudo sense switches will be set.

RESET,value specifies which pseudo sense switches are to be reset. The "value" may be from 1 to 6, and more than one value may be specified. If ALL is specified, all pseudo sense switches will be reset.

If a conflict exists between the SET and RESET options, the last setting specified in the command will apply, since the Monitor processes the options in sequence (i.e., left to right).

RUN The RUN control command specifies that a designated program (or the program most recently formed by the loader) is to be executed, provided that the execution error severity level (see XSL, below) has not been exceeded by the program (i.e., the load module).

The form of the RUN control command is

```
! RUN [(option)] [,(option)] ... [,(option)]
```

where the options are as follows:

LMN,name ,account ,password specifies the name (account number and associated password, if any) of the load module that is to be executed. If this option is omitted, the job's most recently formed load module will be executed.

START,address specifies the location at which program execution is to begin. The "address" may be either an external definition (optionally followed by a hexadecimal addend value) or a signed absolute hexadecimal address.

BIAS,address specifies the address to which the load module is to be relocated. The "address" must be on an absolute hexadecimal page boundary.

XSL,value specifies the highest error severity level that is to be tolerated in accepting a program for execution (see Table 5). If XSL is omitted, a value of 4 is assumed.

INT,address [,DISABLE] specifies the absolute hexadecimal interrupt location to be associated with the real-time program. If the named program has been designated as a resident foreground task, it will be loaded into its assigned area. If the named program has been designated as a non-resident foreground task, it will be loaded when the respective interrupt occurs. If DISABLE is specified, the interrupt will be armed and disabled; otherwise, it will be armed and enabled. If INT is specified, neither TIME nor CLOCK (see below) may be specified in the RUN command.

TIME,value specifies a time interval (after a real-time program has been loaded and the timer started), in hundredths of a second, following which the real-time program is to be executed. The real-time program will be executed repeatedly at the frequency specified by the designated value. If TIME is specified, neither INT nor CLOCK (see below) may be specified in the RUN command.

CLOCK,address,value [,DISABLE] specifies a signed absolute hexadecimal clock location to be associated with the real-time program. The "value" specifies the time interval, in units of the clock's resolution, to which the clock is to be set. If DISABLE is specified, the designated clock will be armed and disabled; otherwise, it will be armed and enabled. If CLOCK is specified, neither TIME nor INT (see above) may be specified in the RUN command. If CLOCK is specified, DIRECT (see below) is assumed also; but, if a specified register block is to be used, then the DIRECT option must be specified explicitly.

DIRECT [,rblock] specifies that the real-time program is to be entered directly each time the associated interrupt or clock becomes active. The foreground routine also must have been declared RESIDENT and MASTER at System Generation time. The program will be entered in the master mode and is responsible for saving and restoring any machine environment it changes (and returning control to the point at which the interrupt occurred).

The sub-option "rblock" specifies a register block to be associated with the task. If DIRECT is not specified (for a foreground task), the task will be entered through the Monitor's interrupt service routine and the Monitor will automatically save and restore the machine environment and return address. DIRECT may be specified only if CLOCK or INT (see above) is also specified in the RUN command.

Example:

```
! RUN (INT,6D),(XSL,3)
```

This example specifies that the program to be executed is a foreground program to be initiated on the occurrence of an interrupt at location 6D. It also specifies that the program is not to be executed if the load module contains errors of a greater severity level than 3. Since neither START nor LMN is specified, the most recently encountered starting address in the most recently formed load module will be used.

DEBUG CONTROL COMMANDS

Debug control commands described under this heading may be used only following a RUN command. With the exception of the IF, AND, and OR control commands, they may appear in any random sequence after the RUN command for the program to which they apply. Only one page is reserved for debug control command FPTs generated by the program loader. If this limit is exceeded, the error message "TOO MANY DEBUGS" is listed.

Memory dumps performed in response to debug control commands may be either conditional or unconditional. Conditional dumps are dependent on the outcome of a program's execution (e.g., whether errors occurred during program execution). The memory dump lists the current PSD and registers, followed by the requested memory areas.

A location appearing in a debug command may be specified as a hexadecimal address, an external definition, or an object module name. Addresses relative to external definitions consist of the label of the external definition optionally followed by a signed hexadecimal addend value (e.g., LOC+B).

All dynamic debug commands (i.e., SNAP, SNAPC, IF, AND, OR, COUNT) cause the specified instruction to be replaced by a Monitor call. The replaced instruction will be executed after the specified action takes place.

All debug output is listed on the DO device. Debug control commands PMD and SNAP (see below) may be used to dump only those areas within the limits of the load module, since any pages obtained by means of M:GP or M:GCP procedure calls are not known to the Monitor at the time that the program is loaded.

POSTMORTEM DUMP CONTROL COMMANDS

A postmortem dump control command requests the Monitor to dump a selected area of memory. Such a dump is termed "postmortem" because, in general, it is performed after the program has been executed or terminated due to error (i.e., "errored"). If an error is detected during program execution, the Monitor lists an appropriate error message on the LL device, in addition to listing the dump output on the DO device.

Any number of separate program areas may be specified for a program, in one or more postmortem dump commands. If no program areas are specified, all areas having a protection code of 00 will be dumped. If a single job includes several programs to be loaded and executed separately, each such program may have one or more associated postmortem dump control commands.

There are two types of postmortem dump commands (see below), neither of which may be used prior to the RUN command for the program to which they apply.

PMD A PMD control command causes a specified dump to be output only if an error occurred during program execution or if the program has returned control to the Monitor via an ERR or XXX return (see Chapter 4).

The form of the PMD control command is

```
! PMD[,segment]  [(from,to)] . . . [,(from,to] [,(pp)]
```

where

segment specifies the name of an overlay segment (valid only if an OVERLAY control command was used in loading the program). If the segment name is omitted, the root (of an overlay program) is assumed.

from,to specifies the location of the beginning (from) and end (to) of an area to be dumped. Either "from" or "to" may be expressed as a relative hexadecimal location (i.e., an external definition followed by an optional hexadecimal addend value) or a positive (preceded by a "+" character) absolute hexadecimal address.

pp specifies the memory-access class that is to be dumped (see Appendix A).

Examples:

```
! PMD
```

This example specifies that the data areas of the program (or of the root portion of an overlay program) is to be dumped. It is equivalent to ! PMD (00).

```
! PMD,KALX(10)
```

This example specifies that all areas of overlay segment KALX that have a memory-access code of 10 (i.e., memory access 10) are to be dumped.

```
! PMD,KALX(LOC1+5,LOC2-A)
```

This example specifies that the area to be dumped is that part of overlay segment KALX beginning five words higher than location LOC1 and ending ten words lower than LOC2.

```
! PMD(10),(00)
```

This example specifies that all areas of the program that have a memory-access code of 10 or 00 are to be dumped.

PMDI The PMDI control command causes the specified dump to occur whether or not any errors have been detected.

The PMDI control command is of the form

```
! PMDI[,segment]  specification
```

where

segment specifies the name of an overlay segment (valid only if all areas to be dumped lie within the designated overlay segment).

specification may include any (or all) of the specification options appropriate to a PMD control command (see "PMD", above).

SNAPSHOT CONTROL COMMANDS

A memory snapshot control command requests the Monitor to dump a selected area of memory when a specified instruction is about to be executed. Such a dump is termed a "snapshot" because it provides an instantaneous "picture" of program conditions existing at a particular point in time during program execution.

Any number of separate program areas may be specified in a snapshot control command. All registers are dumped, as is the Program Status Doubleword (PSD).

Note that the location initiating the dump (i.e., causing the dump to take place) must contain an executable instruction that is not altered or replaced during program execution. This requirement is necessary because the Monitor replaces the contents of that location with a branch to the Monitor's snapshot dump routine.

There are two types of snapshot dump commands (see below), neither of which may appear in the job deck prior to the RUN command for the program.

SNAP The SNAP control command requests the Monitor to take an unconditional memory snapshot.

The form of the SNAP control command is

```
! SNAP[,segment]  loc,com[,(from[,to])]. . . [,(from[,to])]
```

where

segment specifies the name of an overlay segment (valid only if the location initiating the dump and also the area to be dumped are located within the specified overlay segment).

loc specifies the location at which the dump is to be initiated. That is, the specified dump is to occur just prior to the execution of the instruction at "loc". Note that "loc" (and either "from" or "to", below) may be expressed as a relative hexadecimal location (i.e., an external definition followed by an optional hexadecimal addend value) or a positive absolute hexadecimal address.

com specifies a string of up to eight alphanumeric comment characters that are to be printed with the dump output. Note that at least one such comment character must be specified in the command.

from specifies the location of the beginning of an area to be dumped.

to specifies the location of the end (i.e., highest core location) of an area to be dumped.

Example:

```
! SNAP TAB,SNAP1,(HERE+14,THERE-1)
```

This example specifies that the area beginning twenty word locations higher (in address) than location HERE and ending one word location lower than THERE is to be dumped just prior to the execution of the instruction at location TAB. The message "SNAP1" is to be printed with the dump.

SNAPC The SNAPC control command requests the Monitor to take a conditional memory snapshot.

The form of the SNAPC control command is

```
! SNAPC[,segment] flag,specification
```

where

segment specifies the name of an overlay segment (see "SNAP", above).

flag specifies the name of the test identifier. It may consist of any character string from one to eight characters in length. Since the Monitor does not associate the flag with the user's program, no confusion with program symbols can arise. The normal state of the flag bit associated with a flag (in a table established and maintained by the Monitor) is the set state. It is set and reset by means of the IF, AND, OR, and COUNT control commands (see below). Unless the flag bit is set, the specified dump cannot take place.

specification must include both of the required parameters of a SNAP control command (i.e. initiating location and comment string) and may also include any or all of the optional specifications (see "SNAP", above).

Example:

```
! SNAPC,NIM AT5,LUP+1,TP33
```

This example specifies that, if flag AT5 is in the set state just prior to the execution of the instruction whose memory address is one (word) greater than that of LUP (in overlay segment NIM), then all general registers and the PSD are to be dumped. If the dump occurs, the message "TP33" is printed with the dump.

IF The IF control command may be used in conjunction with a conditional snapshot command (see SNAPC!, above). It requests the Monitor to make a specified test at a designated location and, if the test condition is found to be true, to set the flag bit associated with the conditional snapshot. If the test condition is found to be false, the flag bit is to be reset by the Monitor.

Since the IF control command may be used in conjunction with other dynamic debug commands (see AND and OR, below), the relative sequence of such commands may affect the performance or inhibition of the dump. It is the user's responsibility to sequence such commands in the order dictated by the logical requirements of the conditional dump.

Note that the instruction at the test location specified in a dynamic debug command must be executed prior to the execution of the instruction at the location that initiates the dump. In the case of an overlay program, both the test location and the dump-initiating instruction must be located in the same segment.

The IF control command is of the form

! IF[,segment] flag,loc,(l_1,x_1[,b_1],r,l_2,x_2[,b_2])

where

segment specifies the name of the overlay segment (if any) that is specified also in the associated SNAPC command (see "SNAPC", above).

flag specifies the name of the test identifier (see "SNAPC", above).

loc specifies the absolute or relative (external definition [± addend]) hexadecimal location at which the test is to take place. That is, the specified test is to occur just prior to the execution of the instruction at "loc".

l_1 and l_2 specify locations that are to be compared as specified by "r" (see below). They may be either absolute or relative and may be indirect ($*l_i$).

x_1 and x_2 specify index registers to be used to modify the addresses specified by l_1 and l_2 (see above), respectively. A zero may be used to specify that indexing is not to be used.

b_1 and b_2 specify the number of bytes to be compared. The permissible values and their meanings are

Value	Meaning
1	Byte 0
2	Halfword 0
4	Full word
8	Doubleword

The values of b_1 and b_2 are normally the same but may be different. If omitted, the value 4 (i.e., full word) is assumed.

r specifies the type of comparison to be made. The permissible values and their meanings are

Value	Meaning
GT	Greater than
LT	Less than
EQ	Equal to
GE	Greater than or equal to
LE	Less than or equal to
NE	Not equal

Example:

```
! PSI-5,5)
```

```
! IF TAU,ETA+1,(RHO+A,4,EQ,;
```

This example specifies that two words in core storage are to be tested to determine whether they are equal (i.e., identical). One of these two words has an address that is ten word locations greater than that of external definition RHO, plus the contents of index register 4. The other word to be compared has an address that is five word locations less than that of external definition PSI, plus the contents of index register 5.

The example also specifies that the test is to occur just prior to the execution of the instruction that is one word location higher than external definition ETA. If the specified test gives a true result, the flag named TAU is to be set; otherwise, the flag is to be reset.

AND The AND control command may be used in conjunction with a conditional snapshot command (see "SNAP", above). It requests the Monitor to make a specified test at a designated location, but only if the flag bit for the associated snapshot is in the set state when the test is to be made. If the test condition is found to be true, the flag bit remains set; otherwise, the flag bit is reset. If the flag bit is in the reset state when the test is to be made, the test is not performed and, unless the flag bit is set as a result of some subsequent command, the associated snapshot does not occur.

The AND control command is of the form

```
! AND[,segment] specification
```

where

segment specifies the name of the overlay segment (if any) that is specified also in the associated SNAPC command (see "SNAPC", above).

specification (see "IF", above).

OR The OR control command may be used in conjunction with a conditional snapshot command (see "SNAPC", above). It requests the Monitor to make a specified test at a designated location, but only if the flag bit for the associated snapshot is in the reset state when the test is to be made. If the test condition is found to be true, the flag bit is set; otherwise, the flag bit remains reset and, unless the flag bit is set as a result of some subsequent command, the associated snapshot does not occur.

The OR control command is of the form

```
! OR [,segment] specification
```

where

segment specifies the name of the overlay segment (if any) that is specified also in the associated SNAPC command (see "SNAPC", above).

specification (see "IF", above).

COUNT The COUNT control command allows the user to specify an iteration range (and steps within that range) in which a designated test identifier (i.e., a flag for a snapshot dump) will be set. A separate internal counter is established by the Monitor for each COUNT command and the count is incremented by one whenever (i.e., just before) an instruction at a specified location is executed. The iteration count is then tested to determine whether the flag for the specified dump will be set or reset. COUNT operates independently of any OR, IF, or AND commands.

The flag for the designated dump will be set if the current count is within the range of the specified start and end count, and if the quotient "(count-start)/step" is an integer. Otherwise, the flag will be reset.

The COUNT control command is of the form

```
! COUNT[,segment]  flag,loc,start,end,step
```

where

segment specifies the name of the overlay segment (if any) that is specified also in the associated SNAPC command (see "SNAPC", above).

flag specifies the name of the test identifier (see "SNAPC", above).

loc specifies the absolute or relative (external definition [±addend]) hexadecimal location at which the count is to be incremented by one.

start specifies the decimal count at which the testing of the count is to begin. When the count equals "start", the flag is set (even if "start" is equal to zero).

end specifies the decimal count at which the incrementing of the count is to cease. A maximum value of 2,147,483,647 may be specified.

step specifies the decimal count increment that determines the intervals (within the range designated by "start" and "end") at which dumps are to be taken. Both "step" and "start" must be less than "end".

Example:

```
! COUNT  FLG26,HERE+6,1,10,1
```

This example specifies that the name of the flag to be set is FLG26, the count is to be incremented by one just prior to executing the instruction located six word locations higher than external definition HERE, the count range within which the count is to be tested (to determine if "count-start/step" is an integer) is from 1 to 10, and the flag is to be set whenever the count is incremented by one. Note that, in this example, the flag is not set until the count reaches 2.

INPUT CONTROL COMMANDS

Note that input control commands must not have any spaces between the exclamation character and the mnemonic.

BIN The BIN control command informs the Monitor that the information to follow will be in binary. The termination of the binary information is specified by a BCD control command (see "BCD", below).

The form of the BIN control command is

```
!BIN
```

BCD The BCD control command is used as a terminator for binary input, i.e., it causes the Monitor to revert to its "normal" EBCDIC input mode.

The form of the BCD control command is

```
!BCD
```

DATA The DATA control command is used to inform the Monitor that a data deck is to follow. The data deck is for use by the executing program.

The DATA control command is of the form

```
!DATA
```

Any debug commands associated with the user's program must precede a DATA control command.

EOD The EOD (i.e., end of data) control command may be used to define data blocks in a data deck. This is accomplished by inserting EOD control commands at the end of each block of data. When an EOD command is encountered, the Monitor returns an abnormal code of 05 in SR3 (if the user has specified an abnormal address for the M:READ procedure).

The EOD control command is of the form

```
!EOD
```

Any number of EOD commands may be used in a job.

FIN The FIN control command is used to inform the Monitor that there are no more jobs to be processed.

The FIN control command is of the form

```
!FIN
```

On encountering a FIN control command, the Monitor outputs a message informing the operator that all current jobs have been completed. The Monitor then enters the wait state, but remains responsive to real-time tasks.

UTILITY CONTROL COMMANDS

The utility control commands described below allow the user to manipulate magnetic tape files.

PFIL The PFIL control command may be used to cause a designated number of physical files on unlabeled tape to be moved (i.e., skipped) in a specified direction. For unlabeled tape, the tape will be positioned after the end-of-file in the direction skipped.

The form of the PFIL control command is

```
! PFIL dcb name[,(BACK)][,(files)]
```

where

dcb name specifies the name of the Monitor DCB associated with the files to be skipped.

BACK specifies that skipping will take place in the reverse direction (the default option is skipping in the forward direction).

files specifies the (decimal) number of files to be skipped. If "files" is not specified, 1 is assumed.

REW The REW control command may be used to cause the tape associated with a specified DCB to be rewound.

The form of the REW control command is

```
! REW dcb name
```

where

dcb name specifies the name of the DCB associated with the tape to be rewound.

WEOF The WEOF control command may be used to cause a physical end-of-file to be written on unlabeled tapes (see "M:WEOF", Chapter 4).

The form of the WEOF control command is

```
! WEOF dcb name
```

where

dcb name specifies the name of the DCB associated with the tape on which the end-of-file is to be written.

3. OPERATOR COMMUNICATION

Operator key-ins and Monitor typeouts provide the means of communication between the operator and the system when operator intervention is necessary to maintain system operation. Monitor typeouts inform the operator of various error or abnormal conditions affecting system operation and, if a key-in from the operator's console (the OC device) is expected, the Monitor outputs the message

!! KEY-IN

on the OC device. All messages from the Monitor to the operator are preceded by two exclamation marks (!!) as shown above.

The common characteristics of all key-ins, whether or not solicited by the Monitor, are:

1. To initiate a key-in, the operator presses the INTERRUPT switch on the processor (CPU) control panel.
2. The operator always types an exclamation mark (!) as the first character, and terminates each key-in by a carriage return (i.e., presses the NEW LINE key).
3. The blank (i.e., space) is used as a field delimiter, and any number of blanks may be used to separate fields.
4. Each BACKSPACE means "delete the previously typed character".
5. To delete an entire message prior to termination of the key-in, the operator presses the EOM (end of message) key.

SYSTEM COMMUNICATION

DATE The DATE (or D) key-in may be used to inform the Monitor of the current date.

The DATE key-in has the form

$$\left\{\begin{array}{l}\text{!DATE}\\ \text{!D}\end{array}\right\}\ \text{month, day, year}$$

where

month specifies the current month (1 ≤ month ≤ 12).

day specifies the current day of the month (1 ≤ day ≤ 31).

year specifies the two least significant digits of the current year (00 ≤ year ≤ 99).

TIME The TIME (or T) key-in may be used to set the system clock to the current time of day.

The TIME key-in has the form

$$\left\{\begin{array}{l}\text{!TIME}\\ \text{!T}\end{array}\right\}\ \text{hour, minute}$$

where

hour specifies the current hour (0 ≤ hour ≤ 23).

minute specifies the current minute (0 ≤ minute ≤ 59).

REQUEST and KEY-IN The operator may request the Monitor to let him dismount a tape from a particular unit. He does this through the REQUEST key-in. The REQUEST key-in has the form

!REQUEST ndd

The Monitor will respond with

!!ndd

followed by nothing if the tape unit is empty, or followed by

DISMOUNT SCRATCH reel number

if a scratch tape is on the unit and may be removed and returned to the scratch pool, or followed by

DISMOUNT AND SAVE reel number

if the tape belongs to someone.

If the unit is in use, the Monitor will type

LATER

REQUEST 7T/9T/MT KEY-IN If the operator wishes the Monitor to tell him on which unit he may mount a tape, he may use the following form of the REQUEST key-in.

$$\text{!REQUEST}\left\{\begin{array}{c}\text{7T}\\ \text{9T}\\ \text{MT}\end{array}\right\}$$

where

7T	requests a 7-track unit.
9T	requests a 9-track unit.
MT	requests any available unit.

PRIORITY The PRIORITY key-in may be used to change the priority status of any system I/O file (i.e., a job in disc storage). An attempt to use a PRIORITY key-in in a system that does not contain a system I/O file directory (i.e., a non-symbiont system) will cause the message

!!KEYERR

to be output on the OC device, since this key-in has no meaning in a non-symbiont system.

The PRIORITY key-in has the form

!PRIORITY system id, yyndd priority

where

system id   specifies the identification number assigned to the job by the Monitor. The system ID is listed on the OC device as each job is entered into the system.

yyndd   specifies the name of the symbolic device for which the file was created.

priority   specifies the new priority number (see "JOB", Chapter 2) for the job. A priority of 0 implies that the job is to be deferred until a higher priority is specified.

WAIT   The WAIT (or W) key-in may be used to discontinue a specified job. However, the discontinued job is not checkpointed by such a key-in.

The WAIT key-in is of the form

$\left\{\begin{array}{l}\text{!WAIT}\\ \text{!W}\end{array}\right\}$ system id

where

system id   is the same as for "PRIORITY", above.

START   The START (or S) key-in may be used to cause a specified job to continue after having been discontinued by a WAIT key-in (see above) or causes the Monitor to end an idle state (the Monitor enters an idle state when the system is first loaded or when a FIN control command is encountered).

The START key-in has the form

$\left\{\begin{array}{l}\text{!START}\\ \text{!S}\end{array}\right\}$ system id

where

system id   is the same as for "PRIORITY".

ERROR   The ERROR (or E) key-in forces an error return to the Monitor (see Chapter 4). The job is terminated immediately and any specified postmortem dump is performed for the program being executed.

The ERROR key-in has the form

$\left\{\begin{array}{l}\text{!ERROR}\\ \text{!E}\end{array}\right\}$ system id

where

system id   is the same as for "PRIORITY".

The Monitor responds with the message

!!JOB id ERRORED AFTER xxxx

where

xxxx   is the address of the last instruction executed in the job.

The above message is listed on the LL and OC devices. The PSD and the contents of the general registers are listed on the LL device. If the Monitor is in the wait state when this key-in is input, it is taken out of that state (see "WAIT").

ABORT   The ABORT (or X) key-in forces an abort return to the Monitor (see Chapter 4). The job is terminated immediately and any specified portmortem dump is not performed.

The ABORT key-in has the form

$\left\{\begin{array}{l}\text{!ABORT}\\ \text{!X}\end{array}\right\}$ system id

where

system id   is the same as for "PRIORITY".

The Monitor responds with the message

!!JOB id ABORTED AFTER xxxx

where

xxxx   is the address of the last instruction executed in the job.

The above message is listed on the LL and OC devices. The PSD and the contents of the general registers are listed on the LL device. If the Monitor is in the wait state when this key-in is input, it is taken out of the wait state (see "WAIT).

DELETE   The DELETE key-in may be used to delete a specified system I/O file associated with a specific job. This key-in is applicable to all job files in the file queue, regardless of whether they are input or output files. When a file is deleted, all the disc storage area associated with that file is made available for other use. An attempt to use a DELETE key-in in a system that does not contain a system I/O file directory (i.e., a non-symbiont system) will cause the message

!!KEYERR

to be output on the OC device.

DISPLAY   The DISPLAY key-in may be used to request the Monitor to display (on the OC device) specified information regarding certain aspects of current system operation. Note that if no operational parameter (see below) is specified the entire schedule queue will be displayed. An attempt to use a DISPLAY key-in having none of the options listed below, in a system that does not contain a system I/O file directory (i.e., a non-symbiont system) will cause the message

!!KEYERR

to be output on the OC device.

The DISPLAY key-in has the form

!DISPLAY option

where the option may be any of the following:

DISC specifies that the Monitor is to list the currently available disc storage space as well as those disc sectors that have been locked out from use by the system.

TAPES specifies that the Monitor is to list the device name and status of each of the tape units currently available to the system.

system id specifies that the Monitor is to list any outstanding system I/O files for the specified job.

JOB specifies that the Monitor is to list the system ID of any currently active background job.

WRITELOG The WRITELOG key-in may be used to request the Monitor to output the accounting file on the AL device (normally a card punch).

The WRITELOG key-in has the form

!WRITELOG

The following is the format of records maintained in the accounting log file. One record is created for each !JOB processed. When the operator initiates the key-in !WRITELOG, the entire file is output on the accounting log (M:AL), normally the card punch, at the end of the current job. This also deletes the records. Subsequent jobs create new records, and all records are saved until output.

Total job time is divided into two categories: user time and processor time. User time is time recorded from the time a user's program gets control via a !RUN card until control is returned to the Monitor. All other time is processor time.

These two categories are then subdivided into three more categories each: execution time, I/O time, and overhead time. Execution time is time spent in the slave mode. I/O time is time directly attributable to waiting for user I/O (e.g., unlabeled tape operation with WAIT specified). Overhead time is all other time. Decoding CALs, calling segments of Monitor, calling user overlay elements, blocking/deblocking, waiting for blocking buffers, waiting for symbiont storage or symbiont input, I/O, and clock interrupt time are all included in overhead time.

Temporary disc space used is the sum of the size (in 512-word granules) of all files which were RELeased on closing. This storage is returned to system. Permanent disc space used is the sum of the size of all files which were SAVed by processors or user (even if later released). Accumulated disc space used is the net change in disc storage for this user. This may be negative if the net effect of the job was to release files, or positive if the job created PERManent files.

AL Record Format

Word	Contents (all information in EBCDIC)
0	blanks
1, 2	account number
3	blanks
4, 5, 6	name
7	blanks
8, 9	total job time
10, 11	processor execution time
12, 13	processor I/O time
14, 15	processor overhead time
16, 17	user execution time
18, 19	user I/O time
20, 21	user overhead time
22, 23	# of cards read
24, 25	# of cards punched
26, 27	# of processor pages out
28, 29	# of user pages out
30, 31	# of diagnostic pages out
32, 33	# of scratch tapes used
34, 35	# of save tapes used
36, 37	# of tape reads and writes
38, 39	# of disc reads and writes
40, 41	temporary disc space used (granules)
42, 43	permanent disc space used (granules)
44, 45	accumulated permanent disc space used (granules)

INT The INT key-in may be used to transfer control to the user's console interrupt routine (see "M:INT", Chapter 4). Note that the user's program must set the console interrupt linkage (e.g., by means of a M:INT procedure call).

The INT key-in has the form

!INT system id

where

system id is the same as for "PRIORITY".

SWITCH The SWITCH key-in may be used to change the settings of specified pseudo sense switches.

The SWITCH key-in has the form

!SWITCH system id [, (SET, value, ...)];
[(RESET, value, ...)]

where

system id is the same as for "PRIORITY".

SET, value
RESET, value } (see "SWITCH", Chapter 2).

DIRECT I/O COMMUNICATION

If the Monitor encounters an abnormal condition during an I/O operation, a pertinent message to the operator is output on the OC device. Such a message is of the form

!!name message

where

name is the physical device name (see "STDLB", Chapter 2).

message is the message string informing the operator of the specific condition that has been detected. For example:

ERROR (unable to perform operation)

or

NOT READY (device not ready)

Monitor I/O messages are discussed below, grouped according to the type of device to which they apply.

After correcting the abnormal condition, the operator responds by means of a key-in. The format for an I/O key-in is

!name a

where

name is the physical device name of the device involved in the I/O operation.

a specifies a Monitor-action character (see Table 7).

Table 7. Monitor Actions

a	Monitor Action
C	Continue "as is".
E	Set the appropriate error flag in the DCB associated with the I/O operation, and continue.
L	Continue, but lock out the device from the system after the completion of the current job.
R	Repeat the I/O operation.

CARD READER

If the card reader fails to read properly, or if a validity error occurs, the Monitor outputs the message

!!CRndd ERROR

on the OC device. After correcting the condition, the operator responds with an I/O key-in message. The action character selected (see Table 6) depends on the circumstances.

If a feed check error or a power failure occurs, the Monitor outputs the message

!!CRndd INTLK

on the OC device. If the card in the hopper is damaged, the operator replaces it with a duplicate, presses the RESET button on the card reader, and responds to the Monitor with the key-in

!CRndd, R

In the event of a power failure, the operator presses the RESET button on the card reader and responds to the Monitor with the key-in

!CRndd, R

If the card stacker is full, the hopper empty, or the device is in the manual mode, the Monitor outputs the message

!!CRndd EMPTY

on the OC device. The operator corrects the condition and then simply presses the START button on the card reader, at which point the operation continues.

CARD PUNCH

Instead of outputting an error message when a punch error is first detected, the Monitor punches an error card following each card punched in error, recovers from the error, and then proceeds without operator intervention. The I/O handler attempts to punch a card x times (x = NRA, a DCB variable specified by the user; see Chapter 5) before outputting the message

!!CPndd ERROR

on the OC device. The above message indicates that the card punch is not functioning properly, and the operator should reevaluate the job stack based on this knowledge.

The error cards may be sorted out by the operator or user. They may be sorted visually, as the letters ERR are punched on each one (if a multi-stacker punch is used, no error cards are punched; improperly punched cards are routed to an alternate stacker).

If the input hopper is empty, the stacker is full, or the chip box is full (some machines), or if the device is in the manual mode, the Monitor outputs the message

!!CPndd EMPTY

on the OC device. The operator corrects the condition and simply presses the START button on the card punch.

If a power failure or a feed check error occurs, the Monitor outputs the message

!!CPndd INTLK

on the OC device. If the card in the hopper is damaged, the operator removes it, presses the RESET button on the card punch, and responds to the Monitor with the key-in

!CPndd, R

In the event of a power failure, the operator presses the RESET button on the card punch and responds to the Monitor with the key-in

```
!CPndd,R
```

PRINTER

Whenever a print error is detected, the Monitor outputs the message

```
!!LPndd  ERROR
```

on the OC device. The I/O handler attempts to print a line x times (x = NRA, a DCB variable specified by the user; see Chapter 5) before outputting the above message. The operator's response after correcting the condition depends on the specific device and circumstances.

If the printer is out of paper, the carriage is inoperative, or the device is in the manual mode, the Monitor outputs the message

```
!!LPndd  EMPTY
```

on the OC device. The operator corrects the condition and simply presses the START button on the line printer.

If the line printer power is off, the Monitor outputs the message

```
!!LPndd  INTLK
```

on the OC device. The line printer effects its own recovery in the event of power failure.

PAPER TAPE READER

If an error occurs during the reading of paper tape, the Monitor outputs the message

```
!!PRndd  ERROR
```

on the OC device. After correcting the condition, the operoperator responds with an I/O key-in message. The action character selected (see Table 7) depends on the circumstances.

PAPER TAPE PUNCH

If the paper tape punch is out of paper, the Monitor outputs the message

```
!!PPndd  EMPTY
```

on the OC device. The operator corrects the condition and responds to the Monitor with the key-in

```
!PPndd  INTLK
```

on the OC device. The operator corrects the condition and responds to the Monitor with the key-in

```
!!PPndd,C
```

MAGNETIC TAPE

If an error occurs during the reading or writing of magnetic tape, the Monitor outputs the message

```
!!MTndd  ERROR
```

on the OC device. The I/O handler attempts a recovery x times (x = NRA, a DCB variable; see Chapter 5) before outputting the above message. The operator's response depends on the circumstances.

If a magnetic tape is addressed and there is no power, or the reel is file protected for a write operation, the Monitor will output the message

```
!!MTndd  NOT READY
```

on the OC device. The operator's response depends on the circumstances.

If a magnetic tape is addressed and there is no physical reel, the Monitor will output the message

```
!!MTndd  INTLK
```

on the OC device. The operator corrects the condition by mounting a reel of tape (see below).

The monitor requests the operator to mount a scratch tape by outputting the message

$$\text{!!SCRATCH}\left\{\begin{matrix}7\\9\\M\end{matrix}\right\}\text{T, system id}$$

where

system id   specifies which job requires the use of the scratch tape.

After responding to the above message by mounting a scratch tape, the operator may enter the scratch tape into the system by means of the key-in

```
!SCRATCH  ndd, reel number
```

The Monitor requests the operator to mount a specific volume by outputting the message

$$\text{!!MOUNT}\left\{\begin{matrix}7\\9\\M\end{matrix}\right\}\text{T, reel number, system id}$$

on the OC device.

After responding to the above message by mounting the requested tape, the operator may enter the tape into the system by means of the key-in

```
!MOUNT  ndd, [reel number], system id
```

Upon mounting a reel of labeled magnetic tape, the operator can inform the Monitor of the reel's presence by pressing the RESET, ATTENTION, and START buttons on the front panel of the tape unit. The Monitor's Automatic Volume Recognition (AVR) routine then reads the label record and stores the reel number in a Monitor table (the AVR table). If the reel number is not specified in the label, or it has no label, the message

```
!!AVR ERR
```

is typed on the OC device.

If the tape unit has no ATTENTION button, or if the tape reel has no label, the operator may use a MOUNT key-in (see below) to convey the reel number to the MONITOR. The key-in

```
!MOUNT ndd
```

causes the Monitor to call the AVR routine to read the label record. The key-in shown below causes the Monitor to accept the specified reel number:

```
!MOUNT ndd, reel number, system id
```

where

ndd is as specified in Tables 3 and 4.

reel number specifies the number of the reel that has been mounted.

system id is the same as for "PRIORITY".

The Monitor requests the operator to dismount a specified tape at the conclusion of the job, to write-protect the reel if not protected already, and to save it for future use by outputting the message

```
!!SAVE volume number, reel number, system id
```

on the OC device.

The Monitor informs the operator that a specified tape is being returned to the system for scratch file use by outputting the message

```
!!RELEASE volume number, reel number, system id
```

No intervention by the operator is required.

RAPID ACCESS DISC FILE

Instead of outputting an error message whenever an attempt to perform a disc I/O operation is not successful, the Monitor attempts recovery x times (x = NRA, a DCB parameter; see Chapter 5) before outputting the message

```
!!DC ndd ERROR t,s
```

where

ndd is as specified in Tables 3 and 4.

t is the number of the track on which the operation was attempted.

s is the number of the sector on which the operation was attempted.

The above message indicates that the listed disc area is not available to the system. The operator's response depends on the circumstances. On output, the system selects another disc area and continues writing.

SYMBIONT COMMUNICATION

Although symbionts may be entered as a result of program-initiated I/O requests, any explicit control of symbionts must be done via the operator's console rather than by means of control command cards.

Symbiont key-ins have the form

```
!S name,m
```

where

name specifies the name (see "STDLB" in Chapter 2) of the physical device associated with the symbiont.

m specifies the action to be taken (see Table 8).

Table 8. Symbiont Communication

m	Action to be Taken
C	Continue current symbiont activity for the specified device.
I	Initiate symbiont activity for the specified device.
L	Lock-out symbiont from future activity after this file.
R	Recover to the top of the previous 66-line page (printer only).
X	Terminate this job file.
S	Suspend symbiont activity for the specified device.

Symbionts address the operator via messages on the operator's console. Such messages have the form

```
!!S name, message
```

where

name specifies the name of the physical device associated with the symbiont.

message is the message to the operator (see Table 9).

Table 9. Symbiont Messages

Message	Meaning of Message
ACTIVE	The operator attempted to initiate a symbiont that was already active.
NOT ACTIVE	The operator attempted to suspend a symbiont that was already inactive.
SUSPENDED	The symbiont is temporarily inactive (i.e., has suspended operation).
NOT SUSPENDED	The operator attempted to continue or recover a non-suspended symbiont.
TERMINATED	The symbiont has completed all of its operations and has become inactive.

INSTALLATION COMMUNICATION

SYST The SYST key-in allows the operator to alter standard system assignments (see Table 10) without the necessity of going through System Generation.

The SYST key-in has the form

!SYST label, name

where

label specifies a Monitor operational label to be assigned to a physical device or to another Monitor operational label.

name specifies a physical device name (see "STDLB" in Chapter 2) or a Monitor operational label to which the above operational label is to be assigned.

Table 10. I/O Assignments

Oper. Label	Reference	Permissible Device Types	I/O Function
BI	Binary input	Card reader[†]	Read number of bytes specified (up to 120)
		Magnetic tape	Read number of bytes specified
		Paper tape	Read number of bytes specified
		Disc	Read number of bytes specified
C	Control input	Card reader[†]	Read number of bytes specified (up to 80 in BCD mode, up to 120 in binary mode)
		Typewriter	Read number of bytes specified
		Magnetic tape	Read number of bytes specified
		Paper tape	Read number of bytes specified
CI	Compressed input	Same as BI	
EI	Element input		
SI	Source input	Card reader[†]	Read number of bytes specified (up to 80)
		Magnetic tape	Read number of bytes specified
		Paper tape	Read number of bytes specified
		Typewriter	Read number of bytes specified
		Disc	Read number of bytes specified

[†] Normal assignment

Table 10. I/O Assignments (cont.)

Oper. Label	Reference	Permissible Device Types	I/O Function
BO	Binary output	Card punch[†]	Punch number of bytes specified (up to 120)
		Paper tape	Punch number of bytes specified (up to 120)
		Magnetic tape	Write number of bytes specified (up to 120)
		Disc	Write number of bytes specified (up to 120)
CO	Compressed output	Same as BO	
EO	Element output		
SO	Source output		
AL	Accounting log	Same as DO	
DO	Diagnostic output	Line printer[†]	Perform all device format options and print up to one line
		Card punch	Break into 80-character records and punch up to two records.
		Typewriter	Break into carriage-size records, insert carriage returns, and type up to 132 characters
		Magnetic tape	Same as line printer
		Disc	Same as line printer
		Paper tape	Break into 80-character records and punch up to two records
LO	Listing output	Same as DO	
LL	Listing log		
PO	Punch output (binary)	Card punch[†]	Punch number of bytes specified (up to 120)
		Paper tape	Punch number of bytes specified (up to 120)
		Magnetic tape	Write number of bytes specified (up to 120)
	Punch output (EBCDIC)	Line printer[†]	Same as DO
		Magnetic tape	Break into 80-character records and write up to two records
		Paper tape	Same as DO
		Typewriter	Same as DO
		Disc	Same as DO (line printer)
OC	Operator's Console	Typewriter[†]	Write up to 256 bytes

[†] Normal assignment

4. SYSTEM PROCEDURES

Monitor procedures enable the user's symbolic Meta-Symbol program to request a variety of Monitor functions. When a procedure call is encountered during the processing of a program, the processor responds by retrieving a symbolic calling sequence from the procedure library, modifying it according to the parameters specified in the procedure call, and inserting the modified symbolic code into the user's source program (to be translated into object code during a subsequent processing phase).

At execution time, the calling sequence calls an appropriate Monitor routine that, in turn, performs the desired function. In this manual, the Monitor routine called at execution time, as the end result of a procedure call having a command mnemonic of the form M:XYZ, is referred to as Monitor routine XYZ.

When using CII Meta-Symbol, the Batch Processing Monitor Procedure Library is invoked via the directive

```
SYSTEM  BPM
```

This directive defines all of the Monitor procedures discussed in Chapters 4, 5, and 6 of this manual. The 10 070 instructions are invoked by the directive

```
SYSTEM  SIG7[F] [D] [P]
```

Thus, both the SYSTEM BPM and the SYSTEM SIG7 directives should be used. When the SYSTEM BPM directive is processed, a control section is declared for use in generating function parameter lists for Monitor procedures subsequently used.

Often it is desirable to be able to symbolically reference the parameter list associated with a particular procedure. The second element of the label field list is used for this purpose.

For example, assume the user wanted to use the label "RD" to identify the address of the CAL generated for the read function on the C device, and also wanted to use the label "RDFPT" to identify the address of the Function Parameter Table (FPT) for the same function. He could do so by means of the following Meta-Symbol procedure reference in his program:

Label	Command	Argument
RD,RDFPT	M:READ	M:C,(BUF,ALPHA)

Examples:

WR	M:WRITE	etc.
,OPNFPT	M:OPEN	etc.

In the first example, above, the label "WR" identifies the address of the CAL generated for the write function specified by the argument list (represented here by "etc."). In the second example, the label "OPNFPT" identifies the address of the first word of the FPT generated for the OPEN function specified by the argument list (the associated CAL, in this example, is not given a label; hence, the comma preceding "OPNFPT").

GENERAL-PURPOSE PROCEDURES

LOAD OVERLAY SEGMENT

M:SEGLD Monitor routine SEGLD causes a specified overlay segment to be loaded into core storage. If an I/O error occurs in executing the SEGLD routine, or if the specified segment is not found, the job is aborted.

The M:SEGLD procedure call is of the form

```
M:SEGLD    [*]address[,ecb address]
```

where

address specifies the address of the first word of a byte string containing the EBCDIC name of the segment to be loaded. The first byte (i.e., byte 0) of the addressed location must indicate (in binary) the number of characters in the name. An * may be used to indicate indirect addressing.

ecb address specifies the address of a location (in the calling segment or its backward path) containing the word address of the ECB to be associated with the M:SEGLD procedure call.

Calls generated by the M:SEGLD procedure have the form

```
CAL1,8    address
```

where

address points to word 0 or the Function Parameter Table (FPT) shown below.

word 0

X'01'	0———0	ECB address
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1[†]

*	0———————0	Address of byte string
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

[†] A 1 in bit position 0 of word 1 indicates indirect addressing.

LINK TO A LOAD MODULE

M:LINK The Monitor LINK routine causes the calling load module's core information (i.e., program and data, except common dynamic data) to be saved in disc storage. The calling module's core area is made available to the called module. The called module is then loaded into core storage (overlaying the calling program) and control is transferred to it. If there is no transfer address associated with the called module, the job is aborted. The user's temporary and permanent load module libraries are searched for the specified load module. If it is not found or an I/O error occurs in executing the LINK routine, the job is aborted. To effect a return to the calling module, the called module must make use of the Monitor's LDTRC routine (see below).

Any communication between the calling and called load modules must be accomplished through the general registers or common dynamic storage.

The Monitor-assigned file name of the calling program is contained in SR1, allowing the called module to return to the calling location + 1 (via M:LDTRC, returning control to the overlaid program).

The M:LINK procedure call is of the form

M:LINK 'name'[,'account'[,'password']]

where

'name' specifies the EBCDIC image of the name of the load module to which control is to be transferred.

'account' specifies the account from which the load module is to be obtained.

'password' specifies the password associated with the load module.

Calls generated by the M:LINK procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
X'02'	0———————0	A	P

word 1 (first word of 'name')

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
n	α_1	α_2	α_3

⋮

last word of 'name'

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
α_{n-3}	α_{n-2}	α_{n-1}	α_n

first word of 'account'

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
β_1	β_2	β_3	β_4

last word of 'account'

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
β_5	β_6	β_7	β_8

first word of 'password'

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
γ_1	γ_2	γ_3	γ_4

last word of 'password'

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
γ_5	γ_6	γ_7	γ_8

where

A specifies that 'account' is present (A=1) or absent (A=0).

P specifies that 'password' is present (P=1) or absent (P=0).

LOAD AND TRANSFER CONTROL

M:LDTRC Monitor routine LDTRC loads a specified load module (either one that had been partially executed and then saved as the result of an M:LINK procedure call, or else a "new" one), releases the core area used by the calling module, and transfers control to the starting address of the called module. If the called module was a previously saved program, it will be entered at a point immediately following the original M:LINK call, provided that the Monitor-assigned file name of the previously saved program (communicated to the user via SR1) was stored into word 1 of the FPT associated with the M:LDTRC call prior to executing the M:LDTRC call.

The user's temporary and permanent load module libraries are searched for the specified load module. If it is not found or an I/O error occurs in executing the LDTRC routine, the job is aborted.

Any communication from the calling module to the called module must be accomplished through the general registers or common dynamic storage.

The M:LDTRC procedure call is of the form

M:LDTRC 'name'[,'account'[,'password']]

where

'name'
'account' } (see "M:LINK", above).
'password'

Calls generated by the M:LDTRC procedure have the form

CAL1,8   address

where

address   points to word 0 of the FPT shown below.

word 0

X'03'	0——————————0	A	P
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31

The subsequent words of the FPT are of the same form as shown previously for M:LINK.

GET LIMITS

M:GL   The Monitor GL routine may be used to obtain the absolute hexadecimal addresses of common dynamic core storage. The lower limit is returned in SR1 and the upper limit in SR2.

The M:GL procedure call is of the form

M:GL

Calls generated by the M:GL procedure have the form

CAL1,8   address

where

address   points to word 0 of the FPT shown below.

word 0

X'0B'	0——————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

GET COMMON PAGES

M:GCP   The Monitor GCP routine may be called to extend the lower limit of common dynamic storage by a specified number of pages. If the required pages are available at execution time, condition code 1 (i.e., bit 1 of CC) is set to 0. If the required pages are not available, condition code 1 is set to 1 and the number of pages available is returned in SR1. In either case, SR2 contains the address of the first available page (i.e., the address of the lowest common page available).

The M:GCP procedure call is of the form

M:GCP   pages

where

pages   specifies the number of pages by which common dynamic storage is to be extended.

Calls generated by the M:GCP procedure have the form

CAL1,8   address

where

address   points to word 0 of the FPT shown below.

word 0

X'0C'	0————0	Number of pages required
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

FREE COMMON PAGES

M:FCP   The Monitor FCP routine may be called to free a specified number of pages from the lower limit of the current dynamic common storage area. The freed pages are not available for use by the user's program, and any attempt to use such freed pages will result in a trap.

If the specified pages are not part of the user's dynamic storage area, no pages are affected and condition code 1 is set to 1; otherwise, it is set to 0.

The M:FCP procedure call is of the form

M:FCP   pages

where

pages   specifies the number of pages to be freed.

Calls generated by the M:FCP procedure have the form

CAL1,8   address

where

address   points to word 0 of the FPT shown below.

word 0

X'0D'	0————0	Number of pages to be freed
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

GET N PAGES

M:GP   The Monitor GP routine extends the area of core storage that may be used by the user's program. If the specified number of additional pages of memory are available, CC1 (i.e., bit 1 of the CPU's condition code register) is set to a 0; otherwise, CC1 is set to a 1 and the number of available pages is returned in SR1. In any case, SR2 contains the address of the first available page.

The M:GP procedure call is of the form

M:GP   pages

where

pages   specifies the number of additional pages requested.

Calls generated by the M:GP procedure have the form

CAL1,8   address

where

address   points to word 0 of the FPT shown below.

word 0

X'08'	0————0	Number of pages requested
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

FREE N PAGES

M:FP The Monitor FP routine frees a specified number of pages from the high end of the area of core storage that may be used by the user's program. The pages freed are no longer available for use by the user's program, and an attempt by the user's program to access any of the freed pages will cause the job to be aborted.

If the specified pages are not part of the user's dynamic storage area, no pages are affected and condition code 1 is set to 1; otherwise, it is set to 0.

The M:FP procedure call is of the form

M:FP pages

where

pages specifies the number of pages to be freed from use by the user's program.

Calls generated by the M:FP procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

X'09'	0————0	Number of pages to be freed
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

CHECKPOINT JOB

M:CHKPT The Monitor CHKPT routine causes the user's executing program, data area, TCB, and open output DCB files for the current job to be saved on the checkpoint device (i.e., the M:CK device). The IN option (see below) may be used to save open input files as well. Execution continues in the task requesting the checkpoint.

Included with the information checkpointed is a table of restart information containing the job ID, restart address, and magnetic tape and disc storage information. The magnetic tape information defines where tape reels are to be positioned when the job is restarted. The block count (i.e., the number of physical records from the beginning of an unlabeled tape to the current position) is saved with the associated DCB. The key identifying the current record of a labeled tape is saved with the associated DCB. When the job is restarted the block count, or key identifier, is used to restore the tape to the proper position to continue the checkpointed job.

Any number of checkpoints may be made during the execution of a job, but each must be uniquely identified via the user-defined ID (see below). If the user-defined ID of any checkpoint is identical to the ID of a prior checkpoint, the new checkpoint replaces the prior checkpoint.

Any checkpointed program that uses a temporary input file (i.e., one declared temporary via an M:TFILE procedure call) may not be restarted, if that file is to be used, since temporary files are not saved when a job is checkpointed. Because the M:LINK procedure call causes a temporary file (containing the calling load module) to be created, a subsequent checkpoint would cause the loss of the calling module.

The M:CHKPT and M:RESTART (see below) procedure calls generate an external reference to the M:CK DCB that is used by the Monitor in saving the job.

The M:CHKPT procedure call is of the form

M:CHKPT 'id'[,(LOC,address)][,IN]

where

'id' specifies the user-defined ID of the job that is to be checkpointed. The ID must be from 1 to 3 characters in length. The 'id' is appended to the file name to which the M:CK DCB is assigned (if any).

address specifies the address at which execution is to be restarted for the checkpointed job.

IN specifies that open input files are to be saved with the job.

Calls generated by the M:CHKPT procedure have the form

CAL1,4 address

where

address points to word 0 of the FPT shown below.

word 0

X'00'	0 0	IN	Restart address
0 1 2 3 4 5 6 7	8 9 10 11 12 13	14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

Length of ID	First byte of ID	Second byte	Third byte
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

where

IN specifies whether open input files are to be saved (IN = 1) or not saved (IN = 0).

RESTART JOB

M:RESTART The Monitor RESTART routine causes a checkpointed job to be restarted. If the IN option was not specified in the M:CHKPT call, the user is responsible for determining whether any input files that were open during the checkpoint still exist. Temporary files are released by the Monitor when a checkpoint occurs, and therefore may not exist when the job is restarted.

The M:RESTART procedure call is of the form

M:RESTART 'id' [,(LOC,address)]

where

'id' specifies the 1- to 3-character ID of the previously checkpointed job that is to be restarted.

The 'id' is appended to the file name to which the M:CK DCB is assigned (if any).

address specifies the address at which execution is to be restarted. If omitted, execution will commence at the restart address specified in the originating M:CHKPT call.

Calls generated by the M:RESTART procedure have the form

CAL1,4 address

where

address points to word 0 of the FPT shown below.

word 0

X'01'	Restart address
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

Length of ID	First byte of ID	Second byte	Third byte
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

SET MEMORY PROTECTION

M:SMPRT The Monitor SMPRT routine may be called to set the memory protection, for specified pages of core memory, to a specified value (i.e., protection class). If any of the specified pages do not belong to the user's allocated area, condition code 1 is set to 1 and the requested setting is not made; otherwise, condition code 1 is reset to 0.

SMPRT may be used to set access control for systems not using the memory protection feature of the CPU, by simulating access control through use of the write lock feature, in the following manner:

Memory Protection	Corresponding Write Lock
0	User's key
1	3
2	3
3	3

The M:SMPRT procedure call is of the form

M:SMPRT value,from[,to]

where

value specifies the value of the requested memory protection setting (0, 1, 2, or 3)

from specifies the address of the first page to which the specified setting is to apply. If no "to" (see below) is specified, only this page will be affected.

to specifies the address of the last page to which the specified setting is to apply.

Calls generated by the M:SMPRT procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

X'0A'	Address of first page
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

Value	Address of last page
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

GIVE TIME AND DATE

M:TIME The Monitor TIME routine gives the time of day and the current date.

The M:TIME procedure call is of the form

M:TIME address

where

address specifies the address of a four-word block where the time and date are to be stored. The (EBCDIC) byte format of this block is shown below.

word 0

h	h	:	m

word 1

m	ƀ	m	o

word 2

n	ƀ	d	d

word 3

,	'	y	y

where

hh is the hour (00 ≤ hh ≤ 23).

mm is the minute (00 ≤ mm ≤ 59).

mon is the month (standard 3-letter abbr.)

dd is the day (01 ≤ dd ≤ 31).

yy is the year (00 ≤ yy ≤ 99).

Calls generated by the M:TIME procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

X'10'	Address of block to receive time and date
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SET INTERVAL TIMER

M:STIMER The Monitor STIMER routine sets the interval timer with the specified value and specifies what action is to be taken. The interval is to be decremented only when the job issuing the M:STIMER procedure is operating.

When the time expires, the PSD and registers are stored in a block of user's memory on a doubleword boundary. The user's program is entered at "exit address" with register 1 containing the address of the block containing the PSD and general registers. The interrupted program may be reinstated by use of the M:TRTN procedure.

The M:STIMER procedure call is of the form

$$\text{M:STIMER}\quad \left\{\begin{matrix}\text{(MIN,value)}\\ \text{(SEC,value)}\\ \text{(TUN,value)}\end{matrix}\right\}\text{,[[*] exit address]}$$

where

MIN,value specifies (in minutes) the interval to which the timer is to be set.

SEC,value specifies (in seconds) the interval to which the timer is to be set.

TUN,value specifies (in interval timer units) the interval to which the timer is to be set.

exit address specifies the address of a routine to be entered when the specified interval ends. If omitted, the task will resume at the location following the call to STIMER.

Calls generated by the M:STIMER procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'11'	0——0	U	Exit Address
0	1 2 3 4 5 6 7	8 9 10 11 12 13	14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

Interval value
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where

U specifies the type of units represented by the interval (0 means seconds, 1 means minutes, 2 means interval timer units).

TEST INTERVAL TIMER

M:TTIMER The Monitor TTIMER routine causes an indication of the time remaining in the time interval (previously set by the STIMER routine) to be returned to SR1.

The M:TTIMER procedure call is of the form

M:TTIMER [unit] [,CANCEL]

where

unit specifies the units in which the time indication is to be returned to SR1. Unit may be either SEC, MIN, or TUN (see "M:STIMER", above. If omitted, TUN is assumed.

CANCEL specifies that the interval currently in effect is to be canceled. The exit address (see "M:STIMER") is ignored.

Calls generated by the M:TTIMER procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

X'12'	0——0	C	0——0	U
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22	23	24 25 26 27 28 29	30 31

where

C specifies whether the interval in effect is (C = 1) or is not (C = 0) to be concluded.

U specifies the units in which the time indication is to be returned to SR1 (0 means seconds, 1 means minutes, 2 means interval timer units).

TYPE A MESSAGE

M:TYPE The Monitor TYPE routine outputs a specified message to the operator, on the operator's console typewriter.

The M:TYPE procedure call is of the form

M:TYPE (MESS, [*] address)

where

MESS,address specifies the word address of the beginning of the message to be typed. The first byte of the message must specify the number of characters in the message. The message may consist of not more than 255 alphanumeric characters. An optional asterisk may be used to specify indirect addressing.

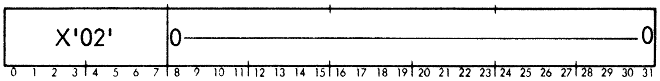
Calls generated by the M:TYPE procedure have the form

CAL1,2 address

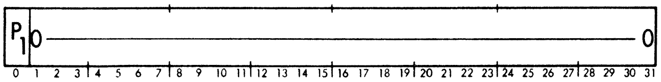
where

address points to word 0 of the FPT shown below.

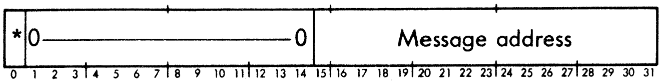
word 0



word 1 (parameter-presence-indicator word)



word 2



P_1 must be equal to 1, indicating that the following parameter word (word 2) is present.

REQUEST A KEY-IN

M:KEYIN The Monitor KEYIN routine types a specified message to the operator and enables the operator's reply to be returned to the user's program.

The M:KEYIN procedure call is of the form

M:KEYIN (MESS, [*] address), (REPLY, [*]address),
(SIZE,value), (ECB, [*] address)

where

MESS, [*] address specifies the word address of the beginning of the message to be output to the operator. The first byte must specify the number of characters in the message. The message may consist of not more than 255 alphanumeric characters.

REPLY, [*] address specifies the word address of the location at which the beginning of the operator's reply is to be stored. The first byte will (automatically) contain the number of characters in the reply.

SIZE,value specifies the maximum number of alphanumeric characters to be accepted from the operator's key-in, and stored.

ECB, [*] address specifies the address of the Event Control Block to be posted when a reply has been received. Bit 0 of the ECB is to be set to a 1 until the reply has been received, then set to a 0.

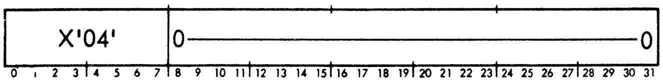
Calls generated by the M:KEYIN procedure have the form

CAL1,2 address

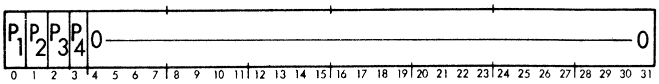
where

address points to word 0 of the FPT shown below.

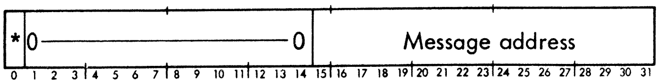
word 0



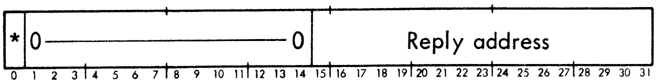
word 1



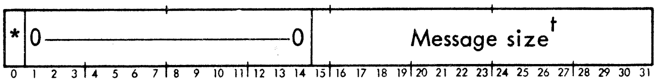
word 2



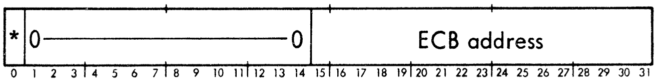
word 3



word 4



word 5



P_1 through P_4 must be equal to 1, indicating that words 2-5 are present.

WRITE TO LISTING LOG

M:PRINT The Monitor PRINT routine outputs a specified message on the listing log (LL device).

The M:PRINT procedure call is of the form

M:PRINT (MESS, [*] address)

where

MESS, [*] address specifies the word address of the location containing the beginning of the message to be output. The first byte must specify the number of characters in the message. The message may consist of not more than 255 alphanumeric characters (132 if LL is assigned to a line printer).

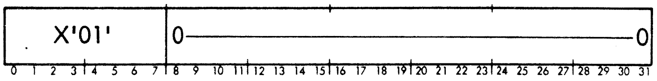
Calls generated by the M:PRINT procedure have the form

CAL1,2 address

where

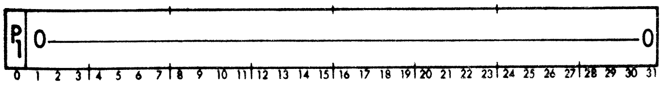
address points to word 0 of the FPT shown below.

word 0

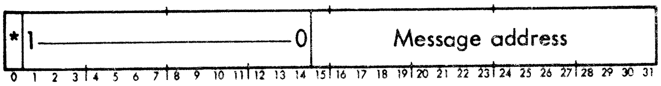


[†] Message size may be indirectly addressed by setting bit 0 of word 4 to a 1. However, this cannot be specified in the procedure call.

word 1



word 2



P_1 must be equal to 1.

SET TRAPS

M:TRAP The Monitor TRAP routine sets and resets the trap conditions. Any trap condition that occurs while in the "trap" state causes control to go to a user's routine; any trap condition that occurs while in the "abort" state causes control to go to a Monitor routine. Maskable traps (i.e., fixed-point and decimal arithmetic) may be masked off so they do not occur, by placing them in the "ignore" state.

Each time the Monitor TRAP routine is entered, the previous contents of the Program Trap Conditions (PTC) become available in SR1 and SR2. The PTC always contains the current trap settings. The first word of the PTC (returned in SR1) indicates which traps are in the "trap" or "abort" state and which maskable traps are in the "ignore" state.

The second word of the PTC (returned in SR2) contains the exit address of the previous trap condition. Using the RESTORE option (see below) and some previous PTC, the trap settings can be restored to a previous setting.

The M:TRAP procedure call is of the form

M:TRAP [exit address] [,(ABORT,traps)] [,(TRAP,traps)]

[,(IGNORE,mask traps)]

or

M:TRAP (RESTORE,ptc address)

where

exit address specifies the relative (external definition [± addend]) or absolute positive hexadecimal location of a user's routine, to handle any traps caused by the TRAP option (see below).

ABORT,trap[,...] specifies the traps to be set to the "abort" state. Any combination of the following (separated by commas) may be specified:

trap	Designated Trap(s)
NAO	Nonallowed operation.
UI	Unimplemented instruction.
WDOG	Watchdog timer (foreground only).
PS	Push-down stack limit.
FX	Fixed-point arithmetic.
FP	Floating-point arithmetic
DEC	Decimal arithmetic.
ALL	All of the above.

TRAP,trap[,...] specifies the traps to be set to the "trap" state. Any combination of the above may be specified (ABORT is assumed by default).

IGNORE,mask traps specifies which maskable traps are to be set to the "ignore" state. Any of the following may be specified:

mask traps	Designated Trap(s)
FX	Fixed-point arithmetic.
DEC	Decimal arithmetic.
BOTH	Both of the above.

RESTORE,ptc address specifies the relative location (external definition ±addend) or absolute positive hexadecimal location of a previous PTC.

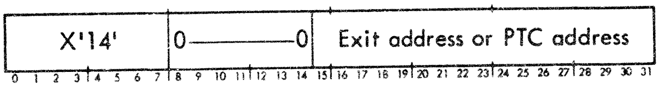
Calls generated by the M:TRAP procedure have the form

CAL1,8 address

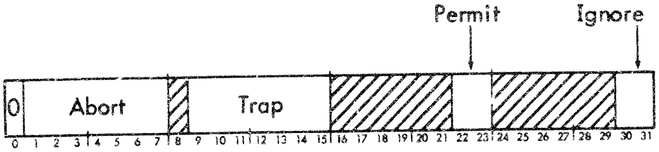
where

address points to word 0 of the FPT shown below.

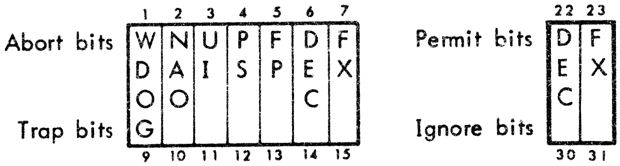
word 0



word 1



where the control bits are as shown below:



When a user's trap routine is to be executed, due to the occurrence of a trap condition for a set trap, the Program Status Doubleword (PSD), current general registers, and trap location are stored in that order into a 19-word block of temporary storage, on a doubleword boundary, and a pointer to word 0 of that block is placed in current general register 1. When a user's trap routine is entered, the condition codes are those loaded by the execution trap. The trap return function (see "M:TRTN", below) can be used to return to the trapped program. If the PSD for the trapped program is to be changed, the user must change the PSD (in temporary storage) before control is returned to the trapped program.

CONNECT CONSOLE INTERRUPT

M:INT The Monitor INT routine may be called to connect a console interrupt (key-in addressing the program) to a user's program, allowing execution of the program to be controlled from the operator's console. When control is given to the INT routine, the PSD and general registers are stored into a 19-word block of user's memory (on a doubleword boundary) and a pointer to word 0 of that block is placed in current general register 1. When a user's interrupt routine is entered, the condition codes are those loaded by the execution of the interrupt. The TRTN routine (see "M:TRTN", below) may be used to restore control from a console interrupt.

The M:INT procedure call is of the form

M:INT address

where

address specifies the location of the entry to the user's console interrupt routine.

Calls generated by the M:INT procedure have the form

CAL1,8 address

where

address points to word 0 of the FPT shown below.

word 0

X'0E'	0————0	Address of interrupt routine
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

The occurrence of a trap condition for one on which the corresponding trap control bit has been set causes the Program Status Doubleword (PSD), current general registers, and trap location to be stored (in that order) into a 19-word block of temporary storage (see Chapter 7). This block is located on a doubleword boundary, and a pointer to word 0 of the block is placed in current general register 1.

The user's trap routine is then entered at the exit address specified in the procedure call. The condition codes are those loaded by the execution of the trap. The Monitor trap return routine (see "M:TRTN", below) may be used to return to the trapped program. If the PSD for the trapped program is to be changed, the user must change the PSD (in temporary storage) before control is returned to the trapped program.

EXIT FROM TRAP OR TIMER ROUTINE

M:TRTN The Monitor TRTN routine restores control to a trapped program. The PSD and contents of the general registers at the time the trap occurred are loaded from the 19-word block of temporary storage used by the most recent trap occurrence.

The M:TRTN procedure call is of the form

M:TRTN

Calls generated by the M:TRTN procedure have the form

CAL1,9 5

No FPT is required.

SIMULATE A TRAP

M:STRAP The Monitor STRAP routine simulates the occurrence of a trap condition specified by a block in temporary storage (at the top of the user's temp stack). If the SPD for the user's temp stack contains an even-numbered address, the user's information must be entered into the stack as follows:

1. Word 0 (the word pointed to by the SPD) must contain 0.
2. Words 1 and 2 must contain the PSD for the simulated trap occurrence.
3. Words 3 through 18 must simulate the contents of the registers that would have been active if the trap had occurred.
4. Word 19 must contain the address of the simulated trap.

If the SPD for the user's temp stack contains an odd-numbered address, the user's information must be entered into the stack as follows:

1. Word 0 (the word pointed to by the SPD) must contain 0.
2. Word 1 must contain -1 (i.e., X'FFFFFFFF').
3. Words 2 and 3 must contain the PSD for the simulated trap occurrence.
4. Words 4 through 19 must simulate the contents of the registers that would have been active if the trap had occurred.
5. Word 20 must contain the address of the simulated trap.

The traps that may be simulated are locations X'40' through X'46' and X'48' through X'4B'.

The M:STRAP procedure call is of the form

M:STRAP

Calls generated by the M:STRAP procedure have the form

CAL1,9 4

No FPT is required.

EXITS TO THE MONITOR

To enable the Monitor to provide continuous system operation, control of the system must be returned to the Monitor by each user's program when it has terminated execution of its operations (for any reason). The Monitor provides three return routines by which the user's program may relinquish control. The user's program must select the return that is appropriate for the circumstances under which the program terminates.

M:EXIT An EXIT return should be used when the user's program has completed its operations in a normal manner. When control is returned via the EXIT routine, the Monitor performs any PMDI dumps that have been specified for the program and then proceeds to the next control command.

The M:EXIT procedure call is of the form

M:EXIT

Calls generated by the M:EXIT procedure have the form

CAL1,9 1

No FPT is required.

M:ERR An ERR return is used when an error has occurred during program execution and the user wants the Monitor to discontinue execution of the current program and proceed to the next task (if any) in the job, after performing any specified postmortem dumps. The Monitor outputs the message

!! JOB id ERRORED BY USER AT xxxxx

where

xxxxx is the address of the last instruction executed in the program.

This message plus the contents of the current register block and Program Status Doubleword (PSD) are listed on the LL device. The PSD contains the address of the last instruction executed in the errored program.

The Monitor also lists the message

!! JOB id ERRORED

on the operator's console.

The M:ERR procedure call is of the form

M:ERR

Calls generated by the M:ERR procedure have the form

CAL1,9 2

No FPT is required.

M:XXX The XXX (abort) return is used when an irrecoverable error has occurred in the execution of the user's program, and the job is to be aborted. When a job is aborted, the Monitor lists the message

!! JOB id ABORTED BY USER AT xxxxx

where

xxxxx is the address of the last instruction executed in the program.

This message plus the contents of the current register block and Program Status Doubleword (PSD) are listed on the **LL** device. The PSD contains the address of the last instruction executed in the aborted program.

The Monitor also lists the message

!! JOB id ABORTED

on the operator's console.

The M:XXX procedure call is of the form

M:XXX

When a job is aborted, any specified postmortem dumps are performed, but no further control commands are honored until a JOB or FIN control command is encountered.

Calls generated by the M:XXX procedure have the form

CAL1,9 3

No FPT is required.

FOREGROUND PROCEDURES

TRIGGER FOREGROUND INTERRUPT

M:TRIGGER By causing the appropriate interrupt to occur, foreground tasks may call the Monitor TRIGGER routine to cause the initiation of other foreground tasks.

The M:TRIGGER procedure call is of the form

M:TRIGGER $\left(\begin{Bmatrix}\text{CLOCK,address}\\ \text{INT,address}\end{Bmatrix}\right)$

where

CLOCK,address specifies the location of a clock interrupt to be triggered.

INT,address specifies the location of an external interrupt to be triggered.

Calls generated by the M:TRIGGER procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'00'	0 0 0	I	0 0 0	Address of interrupt
0 1 2 3 4 5 6 7	8 9 10	11	12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where

I specifies that the interrupt is a clock interrupt (I = 0) or an external interrupt (I = 1).

ARM INTERRUPT

M:ARM The Monitor ARM routine may be called to arm and connect a foreground task to a specific interrupt. System facilities may also be assigned to the foreground task.

The M:ARM procedure call is of the form

M:ARM ({CLOCK,address[,value] [,DISABLE]
INT,address[,DISABLE]});

[,(DIRECT[,rblock])] [,(START, [*] address)] ;

where the options shown are defined under "RUN" (see Chapter 2).

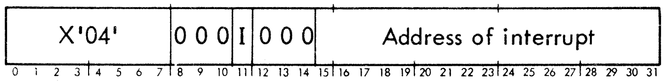
Calls generated by the M:ARM procedure have the form

CAL1,5 address

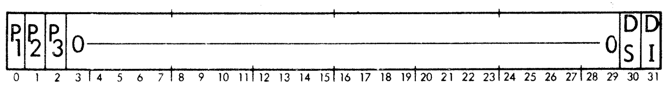
where

address points to word 0 of the FPT shown below.

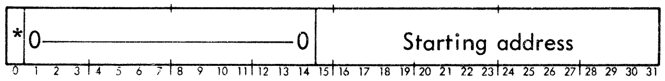
word 0



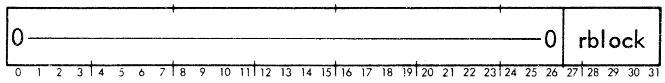
word 1



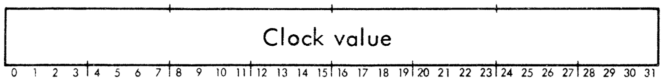
word 2



word 3



word 4



where

I specifies that the interrupt is a clock interrupt (I=0) or an external interrupt (I=1).

DS specifies that the interrupt is to be disabled (DS=1) or not disabled (DS=0)

DI specifies that the interrupt is to be direct (DI=1) or not direct (DI=0).

DISARM INTERRUPT

M:DISARM The Monitor DISARM routine may be called to disarm and connect a foreground task to a specific interrupt. System facilities may also be assigned to the foreground task.

The M:DISARM procedure call is of the form

M:DISARM ({CLOCK,address[,value]
INT,address});

[,START, [*] address)] [,(DIRECT[,rblock])] ;

where the options shown are defined under "RUN" (see Chapter 2).

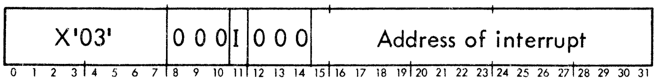
Calls generated by the M:DISARM procedure have the form

CAL1,5 address

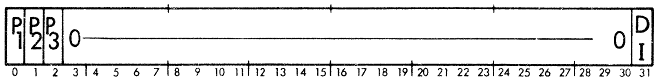
where

address points to word 0 of the FPT shown below.

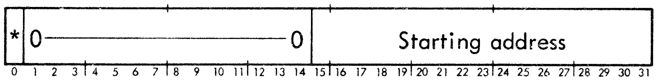
word 0



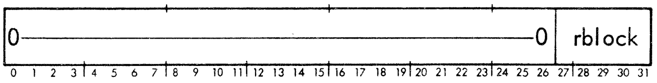
word 1



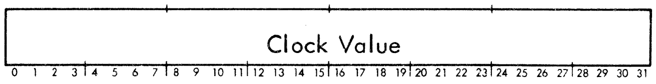
word 2



word 3



word 4



where

I specifies that the interrupt is a clock interrupt (I=0) or an external interrupt (I=1)

DI specifies that the interrupt is to be direct (DI=1) or not direct (DI=0).

SET EXIT CONTROL

M:SXC The Monitor SXC routine may be called to supplant all exits of a foreground program (i.e., all exits to the Monitor). A specified user's routine will handle all exits until a call to RXC (see below) is made.

The M:SXC procedure call is of the form

M:SXC [*]address

where

address specifies the location of a foreground routine that will handle all exits to the Monitor. When an exit occurs, the Monitor will communicate the type of return (see the table below) in SR1 and clear the current exit status.

Code	Type of Exit
X'0'	Normal
X'1'	Trap error
X'2'	I/O error
X'20'	Termination
X'40'	Abnormal
X'80'	Error

Note that an unsolicited abort key-in will abort the task without passing control to the user's program.

Calls generated by the M:SXC procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

* X'0C'	0——————0	Address of exit routine
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

RESET EXIT CONTROL

M:RXC The Monitor RXC routine may be called to restore the user's exits to the Monitor, thereby allowing the Monitor to perform its normal function (see "M:SXC", above).

The M:RXC procedure call is of the form

M:RXC

Calls generated by the M:RXC procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'0D'	0——————————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SAVE BACKGROUND

M:SBACK The Monitor SBACK routine may be called to save the core area of a program that is to be checkpointed (i.e., the executing program). The entire background area is written to the foreground area on the disc and the write locks are set to an access code of 10. The address of the first location in the background area is returned in SR2 and the size, in pages, is returned in SR1. A task may save the background area only once. If the task does not restore the area with a M:RBACK call, the area will be restored by the Monitor when the task exits.

The M:SBACK procedure call is of the form

M:SBACK

Calls generated by the M:SBACK procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'09'	0——————————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

RESTORE BACKGROUND

M:RBACK The Monitor RBACK routine may be called to restore the background area saved by the M:SBACK call. If the task did not save the background area, the task will be aborted.

The M:RBACK procedure call is of the form

M:RBACK

Calls generated by the M:RBACK procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'0A'	0——————————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

CONNECT CAL

M:CAL The Monitor CAL routine (not to be confused with the four CAL machine instructions) may be called to connect a resident foreground program to either CAL3 or CAL4 (machine instructions). The connected program will be entered in the "master" mode and will have the responsibility of saving and restoring any machine environment that it changes (as well as returning program control to the point immediately following the CAL).

The M:CAL procedure call is of the form

M:CAL value, [*] address[,rblock]

where

value specifies the particular CAL that is to initiate the task. The "value" is 3 for CAL3, 4 for CAL4.

address specifies the relative (external definition ±addend) or absolute positive hexadecimal location to which control is to be transferred when the CAL instruction is executed in the user's program.

rblock specifies the number (from 0 through 31) of the register block associated with the CAL.

Calls generated by the M:CAL procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'06'	0———————0	Value

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

1	P 2	0———————0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 2

*	0———————0	Address of connected program

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 3

0———————0	rblock

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 :8 19 20 21 22 23 24 25 26 27 28 29 30 31

DISCONNECT CAL

M:DCAL The Monitor DCAL routine may be called to disconnect a foreground program from a CAL.

The M:DCAL procedure call is of the form

M:DCAL value

where

value (see "M:CAL", above).

Calls generated by the M:DCAL procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'05'	0———————0	Value

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

ENABLE INTERRUPT

M:ENABLE The Monitor ENABLE routine may be called to enable a specified interrupt.

The M:ENABLE procedure call is of the form

M:ENABLE ($\begin{Bmatrix}\text{CLOCK,address}\\ \text{IN1,address}\end{Bmatrix}$)

where the above options are defined under "RUN" (see Chapter 2).

Calls generated by the M:ENABLE procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'02'	0 0 0	I	0 0 0	Address of interrupt

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where

I specifies that the interrupt is a clock interrupt (I=0) or an external interrupt (I=1).

DISABLE INTERRUPT

M:DISABLE The Monitor DISABLE routine may be called to disable a specified interrupt.

The M:DISABLE procedure call is of the form

M:DISABLE ($\begin{Bmatrix}\text{CLOCK,address}\\ \text{INT,address}\end{Bmatrix}$)

where the above options are defined under "RUN" (see Chapter 2).

Calls generated by the M:DISABLE procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'01'	0 0 0	I	0 0 0	Address of interrupt

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where

I specifies that the interrupt is a clock interrupt (I=0) or an external interrupt (I=1).

ENTER MASTER MODE

M:MASTER The Monitor RBACK routine may be called to cause a foreground task operating in the "slave" mode to change its mode of operation to the "master" mode (only if the task has been designated "master" at System Generation).

The M:MASTER procedure call is of the form

M:MASTER

Calls generated by the M:MASTER procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'08'	0————————————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

ENTER SLAVE MODE

M:SLAVE The Monitor SLAVE routine may be called to cause a foreground task operating in the "master" mode to change its mode of operation to the "slave" mode.

The M:SLAVE procedure call is of the form

M:SLAVE

Calls generated by the M:SLAVE procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'07'	0————————————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

TERMINATE TASK

M:TERM The Monitor TERM routine may be called to cause the executing foreground task to be terminated and program control returned to the Monitor. The Monitor will list the following message on the OC device:

! ! TASK name TERMINATED

If the named task was initiated by an interrupt, the interrupt will be cleared and disarmed. If the task is nonresident, its core storage will be released to the Monitor.

The M:TERM procedure call is of the form

M:TERM

Calls generated by the M:TERM procedure have the form

CAL1,5 address

where

address points to word 0 of the FPT shown below.

word 0

X'0B'	0————————————————0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

5. I/O PROCEDURES

The Monitor can perform one operation at a time for each peripheral device. Operations requiring use of the same device are performed in the order requested, and operations using different devices are done simultaneously.

Foreground programs may make use of the Monitor's I/O capabilities or use their own I/O routines. Since all background programs are executed in the slave mode, the privileged instructions required for I/O operations are not directly available to background tasks. Therefore, background programs must call the Monitor's I/O routines to perform I/O functions.

Data Control Blocks (DCBs) are the means by which I/O information is communicated between a user's program and the Monitor. The information required for a particular I/O operation is either contained in the associated DCB or is given in a call. The specific information needed for an I/O operation depends on the organization of the data involved, but is independent of the type of operation to be performed.

As each I/O operation is requested, the associated DCB is set active and is reset appropriately, depending on the outcome of the operation. An attempt to perform another operation through a given DCB while it is active causes the request to be queued, and control to be returned to the user's program if no wait is specified.

System output is usually intended for unit-record devices; therefore, output data is organized sequentially. Systems using the Monitor symbionts may store output data in disc storage until I/O facilities are available, then output it under symbiont control. Monitors that do not allow symbiont operations write output data directly to the unit-record device or to a private file.

The device used for an I/O operation is determined by the contents of the associated DCB when the I/O operation is requested by the executing program.

General registers may not be used as I/O buffers.

I/O procedures are provided for the following I/O functions:

1. File Maintenance

    Create a Data Control Block

    Open a File

    Close a File

    Set Error or Abnormal Address

    Check I/O Completion

    Declare Temporary File

2. Data Record Manipulation

    Read a Data Record

    Write a Data Record

    Release a Data Record

    Delete a Data Record

    Truncate Blocking Buffer

3. File Manipulation

    Position N Records

    Position File

    Close Volume

    Rewind

    Write End of File

4. Special Device

    Set Listing Tabs

    Skip to Top of Form

    Set Number of Printable Lines

    Set Line Spacing

    Specify Direct Formatting

    Specify Vertical Format Control

    Specify Page Count

    Change Output Form

    Change Device Mode or Record Size

    Specify Beginning Column

    Specify Output Header

    Specify Card Punch Sequencing

FILE MAINTENANCE PROCEDURES

CREATE A DATA CONTROL BLOCK

If the user's program is written in CII COBOL or FORTRAN IV, the processor will automatically include all necessary I/O calls and DCBs in the object modules generated for the program. However, if the user's program is written in CII Meta-Symbol, he must provide all necessary I/O procedure calls in his symbolic program. CII Symbol programs must contain explicit code for all I/O calls and user's DCBs, since the Symbol assembler does not process procedure calls.

The user may use Monitor DCBs (see Appendix T) by declaring them as external references in his Symbol or Meta-Symbol program; otherwise, he must create his own DCBs by means of explicit symbolic code (Symbol or Meta-Symbol) or via M:DCB procedure calls (Meta-Symbol only).

All options specified in an M:DCB procedure call or an M:OPEN procedure call (see below) remain in effect only during the execution of the load module containing the call. Thus, if a DCB name is reassigned (via an ASSIGN control command) between the execution of one load module and another, a "new" DCB having that name must be created before that DCB name may be referenced in an I/O call or M:OPEN procedure call.

M:DCB The M:DCB procedure generates nonexecutable code (i.e., it creates only a data area in the user's program) which must have a label. The label is the name by which the DCB is to be referenced.

The M:DCB procedure call is of the form

```
    dcb name  M:DCB  [(option)] ... [,(option)]
```

where

dcb name specifies the name of the user's DCB. The name may consist of from 3 to 31 alphanumeric characters, the first two of which must be "F:". The "dcb name" must previously have been declared a dummy section, via a statement of the form

```
        dcb name DSECT 1
```

The options are as follows:

name (one of the three keyword operands given below).

1. DEVICE, name specifies a system operational label.
2. FILE[, 'name'[, 'account']] specifies a system file directory name of less than 32 characters. If the named file belongs to a different account than that of the current job, the file's account number must be given (either in the M:DCB call or in an ASSIGN control command or M:OPEN call). If the name and account number are both omitted, 16 words are reserved for the name (to be inserted via an ASSIGN control command or M:OPEN call) and 2 words for the account number. If neither FILE nor LABEL (see below) is specified in the M:DCB call, the DCB may not be assigned to any file except a system operational label.
3. LABEL[, 'name'[, 'account']] specifies the name of a file on magnetic tape. If LABEL is specified, an INSN or OUTSN option (see below) must be used to specify the reel containing the file. If the named file belongs to a different account than that of the current job, the file's account number must be given (either in the M:DCB call or in an ASSIGN control command or M:OPEN call). If the name and account number are both omitted, 16 words are reserved for the name (to be inserted via an ASSIGN control command or M:OPEN call) and 2 words for the account number.

org (one of the two file organization types given below).

1. CONSEC specifies that the records in the file are consecutively organized and each record is to be processed in order.

2. KEYED specifies that the location of each record in the file is determined by an explicit identifier (key) that may be used to access the record.

access (one of the two record access means given below).

1. SEQUEN specifies that records in the field are to be accessed in the order in which they appear within the file

2. DIRECT specifies that the next record to be accessed is determined by an explicit identifier (key).

function (one of the four file modes given below).

1. IN specifies the file input mode.

2. OUT specifies the file output mode.

3. INOUT specifies the file input and output mode (i.e., the update mode).

4. OUTIN specifies the file output and input mode (i.e., the scratch mode).

PASS[, 'value'] specifies the password that is to allow access to a classified data file. The password may be from 1 through 8 alphanumeric characters in length. If this option is omitted from the M:DCB procedure call it will not appear in the DCB and, consequently, may not be used in an ASSIGN control command or M:OPEN procedure call referencing the DCB. If PASS is specified but no value given in the M:DCB call, 2 words are reserved for the value (to be inserted via an ASSIGN control command or M:OPEN call).

file (one of the two specifications given below).

1. REL specifies that the Monitor is to allocate temporary space in secondary storage. This option applies only to direct-access files in the OUT or OUTIN mode (see above). Such a file (i.e., one for which REL is specified) can never be saved permanently.

2. SAVE specifies that the file may be included in the system file directory. If the file function (see above) is OUT or OUTIN, the SAVE option must be specified (either in an M:DCB or M:OPEN procedure call, or in an ASSIGN control command), to cause the Monitor to allocate permanent disc storage for the file. When closing such a file, SAVE must again be specified, in the M:CLOSE procedure call, if the file is to be permanently saved in disc storage.

READ[, 'value'] ... [, 'value'] specifies the account numbers of those accounts that may read but not write the file. The value "ALL" may be used to specify that any account may read but not write the file (e.g., READ, ALL). The value "NONE" may be used to specify that no account may read the file. If no value is specified, or if READ is omitted, ALL is assumed by default. The number of accounts explicitly specified in a READ or WRITE specification must not exceed 8. If this option is omitted from the M:DCB procedure call it will not appear in the DCB and, consequently, may not be used in an ASSIGN control command or M:OPEN procedure call. If READ is specified but no values given, 16 words are reserved for Read account number (to be inserted via an ASSIGN control command or M:OPEN call).

WRITE[, 'value'] . . . [, 'value'] specifies the account numbers of those accounts that may have both read and write access to the file. The values "ALL" and "NONE" may be used, as with the READ option (see above); and, if a conflict exists between READ and WRITE specifications, those of the WRITE option take precedence. If no WRITE accounts are specified, NONE is assumed.

If this option is omitted from the M:DCB procedure call it will not appear in the DCB and, consequently, may not be used in an ASSIGN control command or M:OPEN procedure call. If WRITE is specified but no values given, 16 words are reserved for Write account numbers (to be inserted via an ASSIGN control command or M:OPEN call).

INSN [, 'value'] . . . [, 'value'] specifies the serial numbers of the magnetic tape reels that are to be used for file input. These numbers must be ordered in the proper sequence for the file. A maximum of three values may be specified for Monitor DCBs. Serial numbers are from 1 to 4 characters in length. If this option is omitted from the M:DCB procedure call, it will not appear in the DCB and, consequently, may not be used in an ASSIGN control command or M:OPEN procedure call. If INSN is specified but no values given, three words are reserved for serial numbers (to be inserted via an ASSIGN control command or M:OPEN call). Note that the use of INSN has no effect on the file mode.

OUTSN[, 'value'] . . . [, 'value'] specifies the serial numbers of the magnetic tape reels that are to be used for file output. If the output fills the first reel, then the second reel specified will be used, etc. A maximum of three values may be specified for Monitor DCBs. Serial numbers are from 1 to 4 characters in length. If this option is omitted from the M:DCB procedure call, it will not appear in the DCB and, consequently, may not be used in an ASSIGN control command or M:OPEN procedure call. If OUTSN is specified but no values given, three words are reserved for serial numbers (to be inserted via an ASSIGN control command or M:OPEN call). Note that the use of OUTSN, like INSN, does not affect the file mode (i.e., OUTSN will not change the file mode function indicator). If no OUTSN values are specified (either in the M:DCB procedure call or in an ASSIGN control command or M:OPEN call) and the user wishes to create a tape file, the Monitor will request a scratch tape to be mounted. The scratch tape will be saved or released according to the closing specification (see "M:CLOSE" below).

RECL, value specifies the maximum record length, in bytes. The greatest value that may be specified is 32,767. If RECL is not specified, a standard value (appropriate to the type of device used) will apply by default.

TRIES, value specifies the maximum number of recovery tries to be performed for any I/O operation. The greatest value that may be specified is 255. If TRIES is not specified for System DCBs, the standard value established at System Generation time will apply by default. The default for user DCBs is zero.

KEYM, value specifies the maximum length, in bytes, of the keys associated with records within the file. If KEYM is not specified, the value 11 is assumed.

ERR, address specifies the symbolic location of a user's routine that is to be used to analyze any error conditions associated with the makeup of the DCB (see Appendix H).

ABN, address specifies the symbolic address of a user's routine that is to be used to analyze any abnormal conditions associated with the makeup of the DCB.

BTD, value specifies the byte displacement (0-3) in the user's buffer from which I/O is to take place (i.e., at which byte in the buffer the data begins).

VOL, value specifies which tape reel in the list of INSN or OUTSN reels is to be used initially. A value of "1" designates the first reel (in the list), the value "2" designates the second reel, etc. If VOL is omitted, a value of 1 is assumed by default.

NXTF specifies that when the DCB is opened (see "M:OPEN", below) for disc or labeled tape, the Monitor is to access the next file in sequence (following the one most recently accessed via the DCB). If no file name is specified (currently) in the DCB, the first file on the tape or in the user's disc file directory is accessed. If there are no more files available, an abnormal return is executed.

FPARAM, address specifies that the Monitor is to pass the file parameters, in M:OPEN FPT format (see "M:OPEN", below), to the user's program, beginning at the specified "address". The area in the user's program that is to receive the file parameters must be 90 words in length. Only the variable-length parameters are passed to the user's program.

SYNON[, 'file name'] specifies that the "name" given in the FILE option (see above) is to be considered synonymous with the designated "file name". The "file name" must currently apply to the file in disc storage. This option is used to create a synonym for a disc file. It forces the DCB to be open in the update mode. If SYNON is not specified in the M:DCB procedure call, it will not appear in the DCB and, therefore, may not be used in an ASSIGN control command or M:OPEN procedure call referencing the DCB. If SYNON is specified but no value given, 16 words are reserved for the file name (to be inserted via an ASSIGN control command or M:OPEN call).

TLABEL, address specifies the symbolic address of a user's buffer into which a label is to be read, or from which a label is to be written upon opening a disc or tape file. The first byte of the label information must contain the length (i.e., number of bytes) of the buffer.

BUF, address specifies the symbolic address of a buffer that is to be used in the transfer of data.

The following options are device-dependent, and will be ignored by the Monitor in all cases where they are not applicable to the device used.

format (one of the two following specifications).

1. VFC specifies that the first character of each record is a format-control character for printing (see Table 5).
2. NOVFC specifies that the records do not contain format-control characters.

COUNT, tab specifies that a page count is to appear at the top of each page, beginning in the column specified by "tab".

Example:

COUNT,60

The above example specifies that the most significant digit of the page count is to appear in column 60 at the top of each page.

DATA, tab specifies that output is to begin on each page (or card, if EBCDIC) in the column specified by "tab".

SEQ, 'id' specifies that the punched output is to have sequencing in columns 77-80. If 'id' is specified, it will appear in columns 73-76 of the punched output (see ASSIGN, Chapter 2).

LINES, value specifies the number of printable lines per page. The greatest value that may be specified is 32,767. If LINES is not specified, the value established at System Generation time will apply.

SPACE,value[, top] specifies the spacing between lines (value) ana between the top of each page and the first line printed (top). A value of 1 indicates that lines are to be single-spaced. The greatest value that may be specified is 15.

mode (any of the following specifications).

1. BCD specifies that the EBCDIC device mode is to be used.
2. BIN specifies that the binary device mode is to be used.
3. FBCD specifies that FORTRAN BCD conversion is to be used.
4. PACK specifies that the packed binary mode (7-track tape) is to be used.
5. UNPACK specifies that the unpacked binary mode (7-track tape) is to be used.
6. L specifies that a listing type of device is to be used.

If no mode is specified, BCD is assumed.

DRC specifies that the Monitor is not to do special formatting of records on read or write operations.

NODRC specifies that the Monitor is to do record formatting on read or write operations. If neither DRC nor NODRC is specified, NODRC is assumed by default.

TAB, value... [, value] specifies the values of tab stop settings (for an output device). The values must be in ascending order.

HEADER, tab, address specifies that the I/O handler is to output a header (heading) on each page. Tab specifies the column at which the header is to begin. Address specifies the symbolic location of the header; the first byte of the header must contain the number of bytes.

The format and sequence of the machine (data) words comprising a DCB are shown below. DCB words 15 through 21 have two alternative forms. The first form shown applies only to typewriters or line printers. The second form applies to all other types of DCB assignments.

word 0

0-7	8	9	10	11	12-13	14	15	16	17	18	19	20	21	22	23	24-25	26-27	28-31
TTL	MGB	FCI	FCD	WAT	EOP	MOD	NWK	DRC	FBCD	AGV	EGV	EXC	DIR	PUN	VFC	HBTD	UBTD	ASN

word 1

0-7	8-14	15	16	17	18-23	24-31
NRT	FUN	0	DEVF	L	TYPE	DEV/OPLB

word 2

0-7	8-14	15-31
NRA	TYC	BUF

word 3

0-14	15-31
RSZ	ERA

word 4

0-14	15-31
ARS	ABA

word 5

0-1	2-3	4	5	6	7	8-15	16-23	24-27	28-31
FIL1	00	SEQ	SID	TRN	NXTF	0————0	PRI	ORG	ACS

word 6

0-14	15-31
BLK	FLP

word 7

0-7	8-14	15-31
FCN	0———0	QBUF

word 8[†]

0-31
CDA

[†] Two alternative forms of this word are shown.

word 8[t]

0—————————————0	NVA

word 9

CVO or CVI	SBUF1	BUF1

word 10

LVA	KBUF

word 11

COS or CIS	FPARAM

word 12

KEYM	0————0	CLK

word 13

RWS

word 14

CSC	TLB

word 15[t]

TAB1	TAB2	TAB3	TAB4

word 15[t]

BCDA

word 16[t]

TAB5	TAB6	TAB7	TAB8

word 16[t]

IMT	BIUD	MIUD	TBT	BR	EOT	REV	BUF2

word 17[t]

TAB9	TAB10	TAB11	TAB12

word 17[t]

0————0	EXL	EXB

word 18[t]

TAB13	TAB14	TAB15	TAB16

word 18[t]

CBD	KAD

word 19[t]

KEYL	KEY1	KEY2	KEY3

word 19[t]

DSC	SVA	HLC

word 20[t]

CMD	PBD

word 20[t]

HSC	FVA	CVA or SQS

word 21[t]

SID

word 21[t]

ACD	FLD

where

TTL specifies the length of the DCB, in words (a total of 90 words should be reserved for a user's DCB, to allow for expansion of the DCB by the Monitor).

MGB is the Monitor-buffer flag (0 means user's buffer, 1 means Monitor's buffer). The Monitor sets this flag to a 1 when it assigns a buffer to an I/O operation, and sets it to a 0 when it releases the buffer.

FCI is the file-closing flag (0 means no, 1 means yes). The Monitor sets this flag to a 1 when the DCB has been opened previously; otherwise, it is set to a 0.

FCD is the file-closed flag (1 means no, 0 means yes). The Monitor sets this flag to a 0 when a file

[t] Two alternative forms of this word are shown.

associated with the DCB is closed, and sets it to a 1 when the file is opened.

WAT is the wait flag (0 means no wait, 1 means wait). The Monitor sets this flag to a 1 when the system is to wait until the current I/O operation has been completed, and sets it to a 0 when the system is to proceed without waiting.

EOP is the ending-operation indicator (1 means read, 2 means write, 0 means other, e.g., rewind). The Monitor sets this flag to indicate the type of I/O operation most recently performed for the associated file.

MOD is the mode flag (0 means EBCDIC, 1 means binary). The Monitor sets this flag to indicate the device mode to be used in the current I/O operation.

NWK is the new-key flag (0 means existing, 1 means new). The Monitor sets this flag to indicate whether or not the key of the record being written already existed prior to writing the record.

DRC is the device-direct-format indicator (0 means automatic formatting, 1 means no automatic formatting – see Appendix E).

FBCD is the FORTRAN-BCD flag (0 means no conversion, 1 means conversion).

AGV is the abnormal-given flag (0 means no, 1 means yes). It indicates whether an abnormal condition has been detected by the Monitor.

EGV is the event-given flag (0 means no, 1 means yes). It indicates whether the I/O completion type has been communicated to the user's program.

EXC is the exclusive-use flag (0 means exclusive use not wanted, 1 means exclusive use wanted). The Monitor sets this flag to indicate whether or not the current user's program is to have exclusive use of the file or record accessed via this DCB.

DIR is the direction-moved flag (0 means forward, 1 means reverse). The Monitor sets this flag to indicate whether a read operation to be done via this DCB is to take place in the forward or reverse direction.

PUN is the packed/unpacked mode indicator (0 means unpacked, 1 means packed).

VFC is the vertical-format-control flag (0 means no format control, 1 means format control). The Monitor sets this flag to indicate whether or not vertical format control is applicable to operations performed via the DCB.

HBTD is the handler's-byte-displacement indicator (0 means none, 1 means 1 byte, 2 – 2 bytes, 3 – 3 bytes), specifying at which byte in the I/O handler's buffer the data begins (Monitor use only).

UBTD is the user's-byte-displacement indicator (0 means none, 1 means 1 byte, 2 – 2 bytes, 3 – 3 bytes), specifying at which byte in the user's buffer the data begins.

ASN is the assignment-type indicator (0 means null, 1 means FILE, 2 – LABEL, 3 – DEVICE. It is set by the Monitor to indicate the type of assignment currently in effect for this DCB.

NRT indicates the number of recovery tries remaining before the device error message is to be listed.

FUN indicates the file mode function (0 means null, 1 means IN, 2 – OUT, 4 – INOUT, 8 – OUTIN).

DEVF is an indicator defining whether this is a direct device assignment or an operational label assignment (1 means direct device assignment, 0 means operational label).

L is an indicator defining a listing type device (L option specified on !ASSIGN), 1 means listing type device, 0 means nonlisting type.

TYPE is one of the following codes if DEVF equals 1.

Code		Device
1	=	TY
2	=	PR
3	=	PP
4	=	CR
5	=	CP
6	=	LP
7	=	DC
8	=	9T
9	=	7T
A	=	MT

If DEVF equals zero, the TYPE field is meaningless.

DEV is the index of the Monitor device table if DEVF equals zero.

Label	OPLB Value
C	1
OC	2
LO	3
LL	4
DO	5
PO	6
BO	7
LI	8
SI	9
BI	A
SL	B
SO	C
CI	D
CO	E
AL	F
EI	10
EO	11

NRA indicates the number of recovery tries that may be attempted before a device error message to be listed.

TYC indicates the type of completion of an I/O operation (1 means normal, 2 – lost data, 3 – beginning of tape, 5 – end of reel, 8– read error, 9 – write error (see also Appendix H).

BUF contains the address of the user's buffer associated with this DCB.

RSZ indicates the maximum record size, in bytes.

ERA contains the address of the user's routine that will handle error conditions (except for I/O operations).

ARS indicates the actual record size, in bytes, that has been input or output in the current I/O operation using this DCB. If ARS is all 1's RWS (see below) indicates the actual record size.

ABA contains the address of the user's routine that will handle abnormal conditions (except for I/O operations).

FIL1 indicates the file option specified by an M:DCB or M:OPEN procedure call or by an ASSIGN control command (0 means null, 1 – release. 2 – save).

SEQ is the sequence-option flag (0 means no sequence numbering, 1 indicates sequence numbering is desired).

SID is the sequence-ID-given flag (0 means no ID has been specified for punched output, 1 means ID has been specified).

TRN is the truncation flag (0 means the blocking buffer is to be released, 1 means the blocking buffer is not to be released).

NXTF is the next-file indicator (0 means no, 1 means yes). It indicates whether the next file in sequence is to be accessed for the next I/O operation.

PRI indicates the I/O priority code.

ORG is the file-organization indicator or (0 means null, 1 – consecutive, 2 – keyed).

ACS is the file-access indicator (0 means null, 1 – sequential, 2 – direct).

BLK indicates the data block size, in bytes.

FLP is the file-list pointer (for Monitor use).

FCN indicates the current number of outstanding I/O functions for this DCB (i.e., I/O requests that have not yet been satisfied).

QBUF contains the address of the buffer to be used (by the Monitor) in the current I/O operation performed via this DCB.

CDA contains the current device address (i.e., the disc address to be read from or written to).

NVA indicates the number of records to be skipped on magnetic tape.

CVO indicates the volume number of the current tape in a multi-reel output file (e.g., CVO = 2 implies that this is the second tape of the file).

CVI indicates the volume number of the current tape in a multi-reel input file (e.g., CVI = 3 implies that this is the third tape of the file).

SBUF1 indicates the size, in bytes, of the Monitor buffer used to block user's data.

BUF1 contains the address of the Monitor blocking buffer used to block user's data.

LVA indicates the number of printable lines per logical page (for printer or typewriter).

KBUF contains the address of the buffer containing the key most recently used.

COS contains the relative position (in the list of OUTSN numbers) of the serial number of the magnetic tape reel to be used for current file output.

CIS contains the relative position (in the list of INSN numbers) of the serial number of the magnetic tape reel to be used for current file input.

FPARAM contains the receiving address to which file parameters are to be passed; otherwise, it contains 0.

KEYM indicates the maximum length, in bytes, of keys for records in the file associated with this DCB. If KEYM = 0, the maximum length is assumed to be 4 bytes.

CLK contains the cooperative link address (for Monitor use).

RWS indicates the size, in bytes, of the buffer to be used in the current I/O operation performed via this DCB.

CSC indicates the number of the column at which the page count is to begin (for printer or typewriter). The most significant digit of the count will be printed in this column.

TLB contains the address of a user's label that is to be written on a tape file when the file is output via this DCB.

BCDA indicates the current buffer displacement, in records, for the Monitor's blocking buffer, specifying which record the Monitor is processing.

IMT is the image-type flag (0 means account index key, 1 means master file key), used by the Monitor only.

B1UD indicates that the Monitor's primary blocking buffer has (B1UD = 1) or has not (B1UD = 0) been updated.

MIUD indicates that the Monitor's master file index has (MIUD = 1) or has not (MIUD = 0) been updated.

TBT indicates that the Monitor's tape buffer has (TBT = 1) or has not (TBT = 0) been released.

BR indicates that the current record is (BR = 1) or is not (BR = 0) blocked.

EOT indicates that an end-of-tape has (EOT = 1) or has not (EOT = 0) been encountered.

REV indicates that the current block was (REV = 1) or was not (REV = 0) read in the reverse direction.

BUF2 contains the address of the Monitor's secondary buffer.

TAB1-TAB16 indicate listing tabs (i.e., tab stops) for printer or typewriter.

EXL indicates the length, in bytes, of the exclusive-use buffer used to store exclusive-use keys.

EXB contains the address of the exclusive-use buffer.

CBD indicates the current buffer displacement, in bytes, for the Monitor's blocking buffer, specifying at which byte the current record will begin.

KAD contains the address of the current key (or of a dummy key, if the file is not keyed).

DSC indicates the (leftmost) column at which output data is to begin (for a card punch, typewriter, or printer).

SVA indicates the number of lines to be spaced between printed lines and between the top of the page and the first line (1 means single space) on a typewriter or printer.

HLC contains the address of a header (heading) that is to be output (the first byte of the header indicates the length of the header, in bytes).

KEYL indicates the length of the current key, in bytes.

KEY_1 - KEY_3 is the buffer for the key last used for the file associated with this DCB, if KEYM $\le$ 3 (word 12). If KEYM > 3, a dummy is contained in this buffer and the key in KBUF is used to access the file.

HSC indicates the column at which header output is to begin (for a typewriter or printer). The first character of the header will be printed in this column.

FVA indicates the first line on which printing is to begin (for a typewriter or printer).

CVA indicates the current value of the page count (for a typewriter or printer).

SQS indicates the next sequence number to be punched (for a card punch).

CMD is the current master index displacement (in bytes).

PBD is the previous buffer displacement (in bytes), specifying at which byte in the Monitor's blocking buffer the last record begins.

SID indicates the sequence ID for punched card output.

ACD is the account number displacement (in words), specifying displacement from the file list pointer.

FLD is the file name displacement (in words), specifying displacement from the file list pointer.

Entries for any variable-length parameters follow those for the fixed-length parameters already given. Each variable-length entry is preceded by a control word. Byte 0 of each control word contains a unique code number identifying the parameter that follows. Byte 1 contains a code specifying whether the parameter is (code = 01) or is not (code = 00) the last entry of the FPT. Byte 2 specifies the number of significant data words in the entry for the parameter. Byte 3 specifies the total number of FPT words reserved for the entry, not including the control word (i.e., maximum entry length).

The control-word codes in byte 0 identifying each type of variable-length parameter are shown in the table below.

Code	Parameter Type
01	File name (the first byte of which contains the number of characters in the name).
02	Account number.
03	Password.
04	Expiration date.
05	READ account numbers.
06	WRITE account numbers.
07	INSN reel numbers.
08	OUTSN reel numbers.
0B	SYNON name

OPEN A FILE (Reinitialize a DCB)

M:OPEN The Monitor OPEN routine reinitializes specified parameters of a designated DCB; the current values for the remaining entries in the DCB are not altered.

In addition to allowing DCB parameters to be changed prior to an I/O operation, the OPEN routine also sets the DCB's file-closed indicator (FCD) to a 1 (indicating that the DCB is open for use in an I/O operation). No file positioning is done as the result of an Open.

If a READ or WRITE I/O routine is called (see M:READ and M:WRITE, below) while FCD=0, the Monitor stores the call temporarily and calls the OPEN routine automatically. If FCD is still 0 after the OPEN routine has terminated, the requested read or write operation is not executed. The FCD will remain 0 if the information in the DCB is insufficient, inaccurate, or contradictory, and the resulting abnormal error code will be returned in byte 0 of SR3. If the Open is made with no parameters, the existing parameters in the DCB are used.

If the specified DCB is already open (i.e., FCD=1) when the OPEN routine is called, an abnormal condition is signaled (see Appendix H). If the DCB is not open when the OPEN routine is called, the DCB is reinitialized according to the parameters specified in the M:OPEN procedure call.

The M:OPEN procedure call is of the form

M:OPEN [*] dcb name[,(option)]...[,(option)]

where

dcb name specifies the name of the DCB that is to be opened. The options specified in the OPEN call override those previously specified. If no options are specified, the existing ones are used.

The options are as follows:

name (one of the three keyword operands given below).

1. DEVICE, 'name' specifies a system operational label.
2. FILE, 'name'[, 'account'] specifies a system file directory name (no more than 31 characters in length). If the named file belongs to a different account than that of the current job, the file's account number (not exceeding 8 characters) must be given.
3. LABEL, 'name'[, 'account'] specifies the name (not exceeding 31 characters) of a file on magnetic tape. An INSN or OUTSN option (see below) must be used to specify the particular tape(s) containing the file. If the named file belongs to a different account than that of the current job, the file's account number (not exceeding 8 characters) must be given.

org (one of the two file organization types given below).

1. CONSEC specifies that the records in the file are consecutively organized and each record is to be processed in order.
2. KEYED specifies that the location of each record in the file is determined by an explicit identifier(key) that may be used to address the device containing the file.

   If "org" is omitted and DIRECT access is specified (see below), then KEYED is assumed; if neither DIRECT nor "org" are specified, CONSEC is assumed.

access (one of the two record access means given below).

1. SEQUEN specifies that records in the file are to be accessed in the order in which they appear in the file.
2. DIRECT specifies that the next record to be accessed is to be determined by an explicit identifier(key).

   If "access" is omitted and KEYED organization is specified (see above), then DIRECT access is assumed; if KEYED is not specified, SEQUEN is assumed.

function (one of the four file modes given below).

1. IN specifies the file input mode.
2. OUT specifies the file output mode.
3. INOUT specifies the file input and output mode (i.e., the update mode).
4. OUTIN specifies the file output and input mode (i.e., the scratch mode).

PASS, 'value' specifies the password that allows access to a classified data file. The password may be from 1 through 8 alphanumeric characters in length.

file (one of the two specifications given below).

1. REL specifies that, for single-file tapes, the reel is to be released to the Monitor's scratch files. For multifile tapes, no action is to occur if the tape is being used for input; on output, the tape is to be positioned at the beginning of the file(allowing the file to be over written). For direct-access devices (i.e., disc), all allocated direct-access storage for the file is to be released to the Monitor.
2. SAVE specifies that the file is to be included in the system file directory. If the file function (see above) is OUT or OUTIN, the SAVE option must be specified (either in an M:OPEN or M:DCB procedure call or in an ASSIGN control command), to cause the Monitor to allocate permanent disc storage for the file. When closing such a file, SAVE must again be specified – in the M:CLOSE procedure call (see below) – if the file is to be permanently saved in disc storage.

READ, 'value'[, 'value']...[, 'value'] specifies the account numbers of those accounts that may read but not write the file. The value "ALL" may be used to specify that any account may read but not write the file (e.g., READ, ALL). The value "NONE" may be used to specify that no account may read the file. If no value is specified, or if READ is omitted, ALL is assumed by default. The total number of accounts explicitly specified in the READ and WRITE specifications together must not exceed 16 (a maximum of 8 READ and 8 WRITE).

WRITE, 'value'[, 'value'] . . . [, 'value']   specifies the account numbers of those accounts that may have both read and write access to the file. The values "ALL" and "NONE" may be used, as with the READ option (see above); and, if a conflict exists between READ and WRITE specifications, those of the WRITE option take precedence. If no WRITE accounts are specified "NONE" is assumed.

INSN, 'value'[, 'value'] [, 'value']   specifies the serial numbers of magnetic tapes that are to be used for file input. These numbers must be ordered in the proper sequence for the file. A maximum of three values may be specified for Monitor DCBs. Values are 1 to 4 characters in length.

OUTSN, 'value'[, 'value'] [, 'value']   specifies the serial numbers of magnetic tapes that are to be used for file output. If the output fills the first reel, then the second reel specified will be used, etc. A maximum of three values may be specified for Monitor DCBs. Values are 1 to 4 characters in length.

RECL, value   specifies the maximum record length, in bytes. The greatest value that may be specified is 32,767. If RECL is not specified, a standard value (appropriate to the type of device used) will apply by default.

TRIES, value   specifies the maximum number of recovery tries to be performed for any I/O operation. The greatest value that may be specified is 255. If TRIES is not specified, the standard value established at System Generation time will apply by default.

KEYM, value   specifies the maximum length, in bytes, of the keys associated with records within the file. If KEYM is omitted, the value 11 is assumed.

TLABEL, [*]address   specifies the symbolic address of the user's label that is to be written. The first byte of the label information must contain the number of characters in the label when opening an output file, or number of bytes available in the label block when opening an input file. Upon return, the first byte contains the number transmitted.

BUF, [*]address   specifies the symbolic address of a buffer that is to be used in the transfer of data (the buffer may not be a general register).

ERR, [*]address   specifies the symbolic location of a user's routine that is to be used to analayze any error conditions associated with the makeup of the DCB (see Appendix H).

ABN, [*]address   specifies the symbolic address of a user's routine that is to be used to analyze any abnormal conditions associated with the makeup of the DCB.

SYNON,name
NXTF
BTD,value
FPARAM, [*] address
VOL,value

} See "M:DCB", explainea previously in this chapter

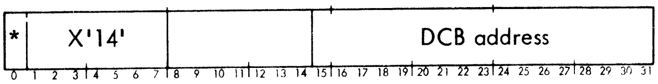
Calls generated by the M:OPEN procedure have the form

CAL1,1   address

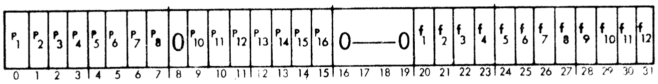
where

address   points to word 0 of the FPT shown below.

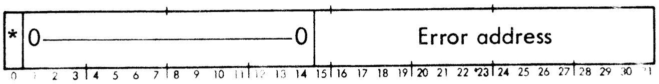
word 0



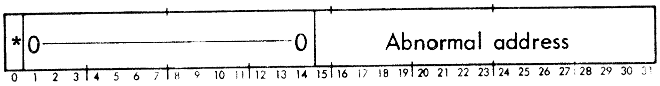
word 1



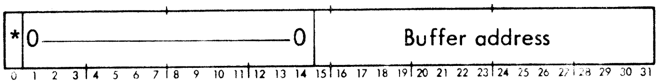
optional



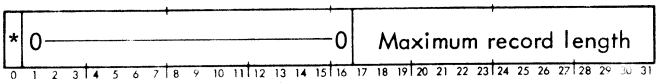
optional



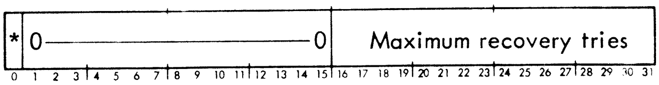
optional



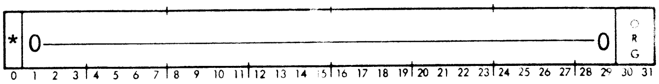
optional



optional



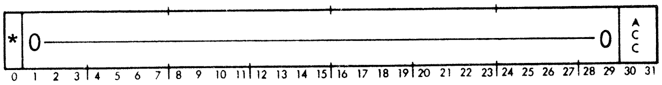
optional



where

ORG   specifies the file organization type (1 means consecutive, 2 means keyed).

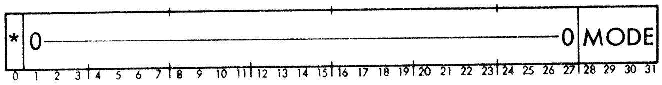
optional



where

ACC specifies the record access method (1 means sequential, 2 means direct).

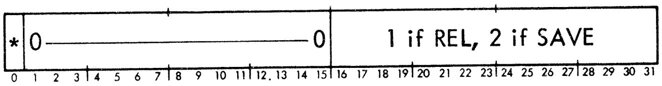
optional



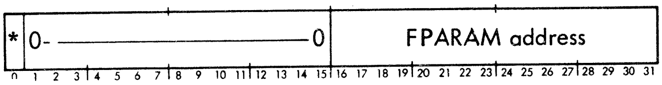
where

MODE specifies the file function mode (1 means IN, 2 – OUT, 4 – INOUT, 8 – OUTIN).

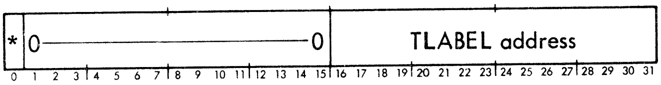
optional



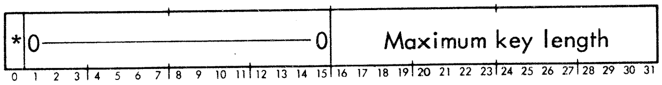
optional



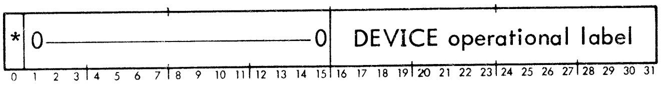
optional



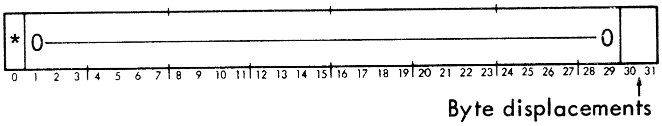
optional



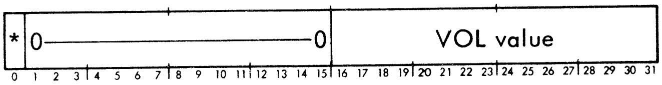
optional



optional



optional



Entries for any variable-length parameters follow those for the fixed-length parameters previously discussed. The variable-length entries are identical in format to those of a DCB. As is the case with all FPTs having optional parameters, the parameter-presence flags (p_1 through p_{16}) indicate which options are present.

Flags f_1 through f_{12} in word 1 of the FPT have the significance indicated below (when f_i = 1).

Flag	Significance (when set to 1)
f_1	A synonym is to be equated to the file name.
f_2	The next file of the account is to be opened. If none are currently open, the first file of the account is to be opened.
f_3	A password is present in the FPT.
f_4	An expiration date is present in the FPT.
f_5	OUTSN serial numbers are present in the FPT.
f_6	INSN serial numbers are present in the FPT.
f_7	WRITE account numbers are present in the FPT.
f_8	READ account numbers are present in the FPT.
f_9	The number of the user's account is present in the FPT.
f_{10}	Not used.
f_{11}	The file is a LABEL type.
f_{12}	The file is a FILE type.

CLOSE A FILE (Terminate I/O Through a DCB)

M:CLOSE The Monitor CLOSE routine terminates and inhibits I/O through a specified DCB, until the DCB is again open (see "M:OPEN", above). It sets the DCB's file-closed indicator to 0, indicating that the DCB is not open for use by an I/O operation. If the DCB being closed is assigned to a card or paper tape punch, an !EOD record is output to the device, indicating an end-of-file (unless the FRM bit is set to 1).

If the DCB is assigned to unlabeled tape and the DRC bit is not set to 1, two end-of-file codes are written and the tape is backspaced prior to the second end-of-file, provided the previous operation was a write operation. (DRC has no effect on labeled tape.)

When a DCB is closed, any special device entries (see M:DEVICE, below) are cleared.

The M:CLOSE procedure call is of the form

M:CLOSE [*] dcb name,(file status)[,(PTL)] [,(REM)]

where

dcb name specifies the name of the DCB that is to be closed.

file status (one of the two status options given below).

1. REL specifies that, for tape files, the reel is to be released for use as a scratch file and a message informing the operator is to be typed. For direct-access devices, all allocated direct-access storage for the file is to be released to the Monitor.

2. SAVE specifies that the current file is to be added to the Monitor's master file, and the file name added to the system file directory. If the file name already exists in the system file directory, the current file will replace the previous one.

If "file status" is omitted for IN or INOUT files, SAVE is assumed; for OUT or OUTIN files, REL is assumed.

PTL specifies that the (labeled magnetic tape) file is to be closed and positioned to the beginning of the file.

REM specifies that the tape is to be rewound and then a message is to be typed requesting the operator to demount the reel. This option is valid for either labeled or unlabeled magnetic tape files.

The options specified in the M:CLOSE procedure call determine whether or not a magnetic tape is to have more than one file. If a reel is to have several files, the SAVE option must be specified in closing each, and the REM option must not be specified until the last file of the reel is to be closed.

An inter-volume mark is written at the end ot each intermediate reel of a multi-reel file when the end-of-reel marker is reached or when a volume is closed (see M:CVOL, below) without closing the last (or only) file on the reel, indicating continuation of the file on another reel.

Calls generated by the M:CLOSE procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'15'	0———0	DCB address

word 1

P1	0———————————0	R	E	0——0

word 2

0———————————0	1 if REL, 2 if SAVE

where

R specifies that the REM option (see above) has (R = 1) or has not (R = 0) been requested.

E specifies that the PTL option (see above) has (E = 1) or has not (E = 0) been requested.

SET ERROR OR ABNORMAL ADDRESS

M:SETDCB The Monitor SETDCB routine allows the user's program to set the error or abnormal address in a designated DCB; the call may be made while the DCB is either open or closed.

The M:SETDCB procedure call is of the form

M:SETDCB [*] dcb name[,(ERR,[*] address)];
[,(ABN, [*] address)]

where the optional parameters are of the same form as those given for ERR and ABN in "M:DCB", above.

Calls generated by the M:SETDCB procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'06'	0———0	DCB address

word 1

P1	P2	0———————————0

optional

*	0———0	Error address

optional

*	0———0	Abnormal address

CHECK I/O COMPLETION

M:CHECK The Monitor CHECK routine checks the completion-type indicator (TYC) of a specified DCB. If TYC $\neq$ 0 and error or abnormal addresses were specified in the procedure call, an appropriate error or abnormal code is returned to the user's program via SR3. If the M:READ or M:WRITE procedure call specified an error or abnormal address, then a normal return to the user's program will be made by the CHECK routine. If I/O is currently active, it will be completed before control is returned to the user's program. If no error address or abnormal address was specified in the procedure call, the Monitor handles any error or abnormal conditions and no error or abnormal code is returned to the user's program. The check applies only to the most recent I/O operation done via the DCB. (See Appendix H.)

The M:CHECK procedure call is of the form

M:CHECK [*] dcb name[,(option)] ... [,(option)]

where

dcb name specifies the name of the DCB to be checked for type of completion.

The options are as follows:

ERR, [*] address specifies the address of a user's routine that will handle error conditions for I/O operations performed via the DCB.

ABN, [*] address specifies the address of a user's routine that will handle abnormal conditions for I/O operations performed via the DCB.

Calls generated by the M:CHECK procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'29'	0——0	DCB address
0	1–7	8–14	15–31

word 1

P1	P2	0——0
0	1	2–31

optional

*	0——0	Error address
0	1–14	15–31

optional

*	0——0	Abnormal address
0	1–14	15–31

DECLARE A TEMPORARY FILE

M:TFILE The Monitor TFILE routine causes a specified DCB to be closed, on return to the user's program, and the associated file to be registered with the Monitor as a scratch file. Error and abnormal addresses may be specified for the DCB. Files declared by means of this call will be released at the end of the job, unless otherwise explicitly released. Thus, a file may be saved between job steps and yet be released on completion of the job.

The M:TFILE procedure call is of the form

M:TFILE [*] dcb name, (TFILE, [*] address);
[,(ERR, [*] address)] [,(ABN, [*] address)]

where

[*] dcb name specifies the name of the DCB associated with the file to be declared temporary.

TFILE, [*] address specifies the address of the name of the file to be declared temporary.

ERR, [*] address (see "M:CHECK" above).

ABN, [*] address (see "M:CHECK" above).

Calls generated by the M:TFILE procedure have the form

CAL1, 1 [*] address[,x]

where

address[,x] points to word 0 of the FPT shown below. The value "x" specifies that indexing is to be used to obtain the effective address (1 ≤ x ≤ 7).

word 0

*	X'0F'	0 0	DCB address
0	1–7	8–14	15–31

word 1

P1	P2	0	1	1	0——0	1	0	1	0	0	0	0
0	1	2	3	4	5–24	25	26	27	28	29	30	31

optional

*	0——0	Error address
0	1–14	15–31

optional

*	0——0	Abnormal address
0	1–14	15–31

required

0——0
0–31

required

*	0——0	File name address
0	1–14	15–31

DATA RECORD MANIPULATION

READ A DATA RECORD

M:READ The Monitor READ routine causes a specified data record to be read into a buffer in core storage. If the record is larger than the specified buffer, part of the record is lost and this fact is communicated to the user's program (see Appendix H). It is not necessary for the user's program to explicitly call the Monitor OPEN routine prior to reading or writing a record, since the Monitor generates such a call automatically if the DCB is not open.

Both EBCDIC and binary decks may be used in the same job, but binary information must be preceded by a BIN control command and must end with a BCD control command if the device is a card reader. On encountering a BIN control command, the Monitor switches the device mode and automatically reads the next record in binary. Subsequent records are also read in binary until a BCD control command is encountered. The Monitor then changes the device mode and automatically reads subsequent records in EBCDIC.

The mode flag (MOD), in the DCB associated with the read operation, is set to a 0 if a record is read in EBCDIC and is set to a 1 if a record is read in binary.

A BCD control command encountered when reading in the EBCDIC mode causes no change in the device mode. When the C device is read, any record having an ! in column 1 (except for a BIN, BCD, or EOD control command) causes a code of 06 to be placed in byte 0 of SR3. The record is placed in the Monitor's control command buffer and, if an attempt is made to read that record again via the same DCB, the job is aborted and the user is notified (via the LL and OC devices) of the reason for aborting the job.

Whenever an EOD control command is encountered (when reading), a code of 05 is returned to the user's program in byte 0 of SR3 if an abnormal address is specified.

The M:READ procedure call is of the form

M:READ [*] dcb name[,(option)] . . . [,(option)]

where

dcb name specifies the name of the DCB to be associated with the read operation.

The options are as follows:

BUF, [*] address specifies the address of the user's buffer into which data is to be read.

SIZE, [*] value specifies the size, in bytes, of the user's buffer. If 0 is specified, a record is skipped. An asterisk may be used to indicate that the "value" is the address of a location containing the buffer size.

ERR, [*] address specifies the address of a user's routine that will handle error conditions for the read operation. (See Appendix H.)

ABN, [*] address specifies the address of a user's routine that will handle abnormal conditions for the read operation. (See Appendix H.)

WAIT specifies that the operation is to be completed before control is returned to the user's program. WAIT is implied if either ERR or ABN is specified. If WAIT is neither specified nor implied, no wait is assumed.

KEY, [*] address specifies the address containing the key (identifier) associated with the record to be read. The first byte of the key contains the length of the key, in bytes. This option is valid only for keyed files.

BTD, value specifies the byte displacement (0 - 3), in the user's buffer, into which data is to be read; i.e., the byte into which the first data byte is to be read.

EXCL specifies that the record to be read may not be accessed by other programs. If EXCL is not specified, NOEXCL is assumed.

NOEXCL specifies that exclusive use of the record to be read is not desired (i.e., the record may be accessed by other programs).

PRI, value specifies the priority of this operation relative to other I/O operations within the job. The "value" may be from 0 to 255 (lowest to highest).

FWD specifies that the record is to be read in the forward direction.

REV specifies that the record is to be read in the reverse direction.

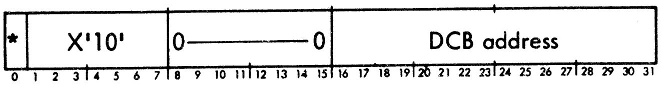
Calls generated by the M:READ procedure have the form

CAL1,1 address

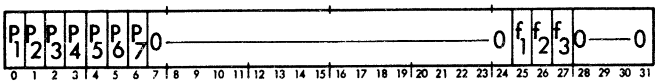
where

address points to word 0 of the FPT shown below.

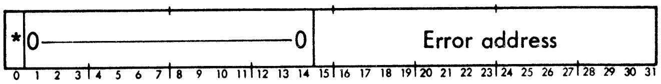
word 0



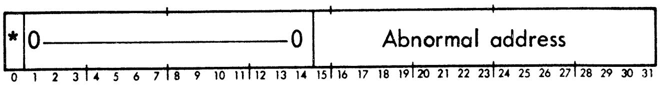
word 1



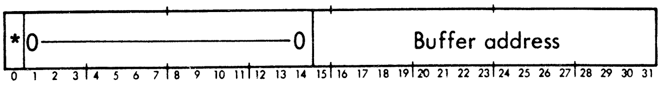
optional



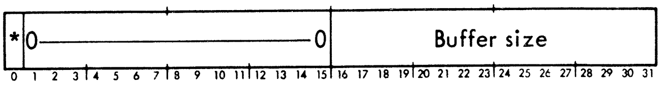
optional



optional



optional



optional

*	0————————0	Key address

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

optional

*	0————————————————0	B T D

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

optional

*	0——————————————0	PRI value

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Flag	Significance
f_1	0 means no exclusive use is desired. 1 means exclusive use is desired.
f_2	0 means read in the forward direction. 1 means read in the reverse direction.
f_3	0 means return control to the user's program immediately. 1 means wait until I/O is complete before returning control to the user's program.

WRITE A DATA RECORD

M:WRITE The Monitor WRITE routine causes a specified data record to be written from a buffer in core storage. The format of the output depends on the type of physical device associated with the DCB.

If the DCB is assigned to a card punch or a paper tape punch, the output for either is handled in a similar manner, except that the Monitor will cause "!BIN" and "!BCD" records to be punched on the card punch where appropriate. For example, if the user's program needs to punch a binary record and the previous record was punched in EBCDIC, an "!BIN" record is punched automatically before the binary record is punched. Similarly, an "!BCD" record is punched automatically before a record is punched in EBCDIC, if the previous record was punched in binary.

On a binary record, a maximum of 120 bytes are punched. On an EBCDIC record, a maximum of 160 bytes are punched, but the data is broken into two records, the first of which contains no more than 80 bytes.

For a line printer, vertical spacing is determined by the first output character if the vertical-format-control flag (VFC) in the associated DCB is set to a 1. A maximum of 132 characters per line may be printed on a line printer.

If the associated DCB is assigned to a typewriter (or to OC), a maximum of 256 characters per write operation is allowed. The user's program must include appropriate carriage return characters in the record to be written. If the DCB is assigned to LO, LL, DO, PO, or BO, a maximum of 160 characters per write operation is allowed; the Monitor will break the output data into two typed lines, the first of which will be 80 characters in length.

The M:WRITE procedure call is of the form

M:WRITE [*] dcb name [,(option)] ... [,(option)]

where

dcb name specifies the name of the DCB to be associated with the write operation.

The options are as follows:

BUF, [*] address specifies the address of the user's buffer from which data is to be read.

SIZE,value specifies the size, in bytes, of the user's buffer. If 0 is specified, the operation is ignored unless records are being written into a keyed file; the key is retained, but the record length is zero.

ERR, [*] address specifies the address of a user's routine that will handle error conditions for the write operation. (See Appendix H.)

ABN, [*] address specifies the address of a user's routine that will handle abnormal conditions for the write operation. (See Appendix H.)

WAIT specifies that the operation is to be completed before control is returned to the user's program. WAIT is implied if either ERR or ABN is specified. If WAIT is neither specified nor implied, no wait is assumed.

ONEWKEY specifies that the NEWKEY option is to be overridden. That is, when updating a record, if the key exists, rewrite the record even though the NEWKEY flag is set to 1; if the key does not exist, write a new key even though the NEWKEY flag is set to 0.

PRI,value (see "M:READ", above).

BTD,value (see "M:READ", above).

KEY, [*] address specifies the address containing the key (identifier) associated with the record to be written. The first byte of the key contains the length of the key, in bytes. This option is valid only for keyed files.

NEWKEY specifies that the KEY (see above) is a new key in the file table. That is, the key of the record to be written must not already exist; if it does exist, an abnormal return is given. (See Appendix H.)

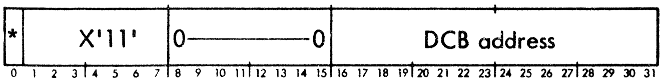
Calls generated by the M:WRITE procedure have the form

CAL1,1 address

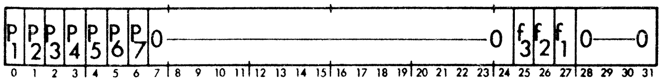
where

address points to word 0 of the FPT shown below.

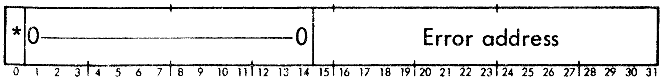
word 0



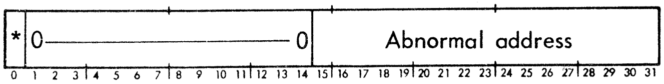
word 1



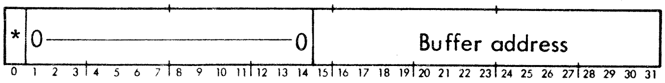
optional



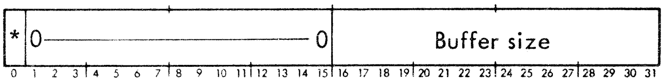
optional



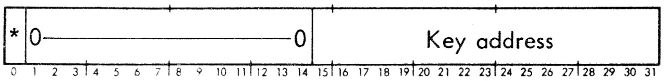
optional



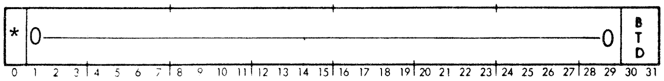
optional



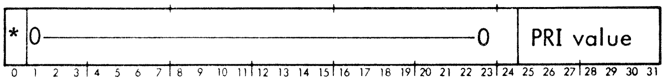
optional



optional



optional



Flag	Significance (when set to 1)
f_1	The WAIT option has been specified.
f_2	The NEWKEY option has been specified.
f_3	The NEWKEY option is to be overridden.

RELEASE A DATA RECORD

M:RELREC The Monitor RELREC routine causes a data record that has been read exclusively (see M:READ, above) to be released for general use.

The M:RFLREC procedure call is of the form

M:RELREC [*] dcb name[,(KEY,address)]

where

dcb name specifies the name of the DCB through which the data record was read.

KEY,address specifies the address of the character string comprising the key that identifies the data record. The first byte of the key specifies the number of bytes in the key. If KEY is omitted, the last record read exclusively through the specified DCB is released.

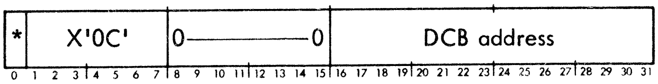
Calls generated by the M:RELREC procedure have the form

CAL1,1 address

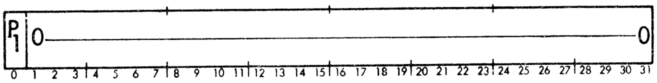
where

address points to word 0 of the following FPT:

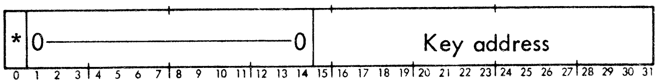
word 0



word 1



optional



DELETE A DATA RECORD

M:DELREC The Monitor DELREC routine causes a data record to be deleted from a file. Note that the INOUT (i.e., update) file mode function must be indicated in the FUN entry of the DCB associated with the file (i.e., FUN = 4).

The M:DELREC procedure call is of the form

M:DELREC [*] dcb name[,(KEY,[*] address)]

where

dcb name specifies the name of the DCB associated with the file containing the record to be deleted.

KEY,address specifies the address of the key that identifies the data record. The first byte of the key specifies the number of bytes in the key. If KEY is omitted, the last record read through the specified DCB is deleted.

Calls generated by the M:DELREC procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'0D'	0——0	DCB address
0	1–7	8–15	16–31

word 1

P1	0———————0
0	1–31

optional

*	0——————0	Key address
0	1–15	16–31

TRUNCATE BLOCKING BUFFER

M:TRUNC The Monitor TRUNC routine causes the Monitor to wait for the completion of any outstanding I/O associated with a specified DCB and then to release the blocking buffer (if any is reserved for the DCB) back to the system for other use. The next READ or WRITE will be assigned a buffer automatically, as needed. This call applies only to files on disc storage or labeled tape.

The M:TRUNC procedure call is of the form

M:TRUNC [*] dcb name

where

dcb name specifies the name of the DCB associated with the blocking buffer to be released.

Calls generated by the M:TRUNC procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'12'	0——0	DCB address
0	1–7	8–15	16–31

word 1

0———————0
0–31

FILE MANIPULATION

POSITION N RECORDS

M:PRECORD The Monitor PRECORD routine causes a specified number of logical records to be skipped in the direction specified.

The M:PRECORD procedure call is of the form

M:PRECORD [*] dcb name, [(N,value)];
[,(option)] [,(option)]

where

dcb name specifies the name of the DCB associated with the file (in disc storage or on labeled or unlabeled magnetic tape).

N,value specifies the number of records to be skipped. The default value is 1.

ABN, [*] address specifies the address of a user's routine to be entered if any of the following abnormal conditions occur: end-of-file, end-of-tape, beginning-of-file, beginning-of-tape. The number of records yet to be skipped is placed in the ARS field of the associated DCB.

FWD specifies that skipping is to take place in the forward direction.

REV specifies that skipping is to take place in the reverse direction.

Calls generated by the M:PRECORD procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'1D'	0——0	DCB address
0	1–7	8–15	16–31

word 1

P1	P2	0——————0	f1	0——0
0	1	2–26	27	28–31

optional

*	0——————0	Number of records to be skipped
0	1–15	16–31

optional

*	0——————0	Abnormal address
0	1–15	16–31

Flag	Significance
f_1	0 means skip in the forward direction. 1 means skip in the reverse direction.

POSITION FILE

M:PFIL The Monitor PFIL routine causes the device associated with a specified DCB to move to the beginning or end of the current file (in disc storage or on labeled or unlabeled magnetic tape).

The M:PFIL procedure call is of the form

M:PFIL [*] dcb name, {(EOF)
{(BOF)}

where

dcb name specifies the name of the DCB associated with the file that is to be positioned.

BOF specifies that the file is to be positioned at its beginning.

EOF specifies that the file is to be positioned at its end (for unlabeled magnetic tape, the file is positioned after the physical EOF mark).

Calls generated by the M:PFIL procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'1C'	0————0	DCB address
0	1–7	8–15	16–31

word 1

0——————————————0	f_1	0——0
0–26	27	28–31

Flag	Significance
f_1	0 means position to the end-of-file. 1 means position to the beginning-of-file.

CLOSE VOLUME

M:CVOL The Monitor CVOL routine causes the Monitor to terminate the reading or writing of data in the magnetic tape reel currently associated with a specified DCB, and to advance to the next reel of the data set.

Unlabeled tapes are positioned at the beginning of the next input reel; output files are positioned at the beginning of a new scratch tape (or output reel, if any). The DCB is closed on the last reel.

For output files on labeled tape, an intra-volume sentinel is written, an alternate tape selected, and a label block written on the alternate tape.

For input tapes, the tape is advanced to the next reel of the data set and the file currently open is located on the next reel.

The M:CVOL procedure call is of the form

M:CVOL [*]dcb name

where

dcb name specifies the name of the DCB associated with the volume to be closed.

Calls generated by the M:CVOL procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'03'	0————0	DCB address
0	1–7	8–15	16–31

REWIND

M:REW The Monitor REW routine causes the tape reel associated with a specified DCB to be rewound if the DCB is open. If the DCB is closed but had been opened previously to a labeled or unlabeled tape, the tape is positioned to the beginning-of-file (labeled tape) or the beginning-of-tape (unlabeled tape).

The M:REW procedure call is of the form

M:REW [*]dcb name

where

dcb name specifies the name of the DCB associated with the file that is to be rewound.

Calls generated by the M:REW procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'01'	0————0	DCB address
0	1–7	8–15	16–31

WRITE END-OF-FILE

M:WEOF The Monitor WEOF routine causes an end-of-file to be written on the unlabeled tape associated with a specified DCB, or !EOD to be output to card punch or paper tape punch, and a top-of-form to be output to the line printer.

The M:WEOF procedure call is of the form

M:WEOF [*]dcb name

where

dcb name specifies the name of the DCB associated with the tape on which the end-of-file is to be written.

Calls generated by the M:WEOF procedure have the form

CAL1,1 address

where

address points to the word 0 of the FPT shown below.

word 0

*	X'02'	0————0	DCB address
0	1–7	8–15	16–31

SPECIAL DEVICE PROCEDURES

M:DEVICE The Monitor DEVICE routine is capable of performing a variety of functions. The function performed is determined by the keyword specified in the procedure call. In all cases where the M:DEVICE call is not compatible with the device associated with the specified DCB, the call is ignored and no error or abnormal return is given. (The DCB must be ASSIGNed to a DEVICE file.)

SET LISTING TABS

This call allows the user's program to set listing tabs for designated columns of data output listed via a specified DCB.

The procedure call is of the form

M:DEVICE [*] dcb name, (TAB, value[, value] . . .)

where

dcb name specifies the name of the DCB associated with the device on which data is to be listed.

TAB, value[, value] specifies the values (column numbers) of desired tab positions. As many as 16 tab values may be specified. The tab values are stored in the TAB fields of the specified DCB in the sequence in which they are specified in the procedure call. A value of 0 specified at TAB_i causes TAB_i through TAB_{16} to be set to 0, indicating null tabs.

Calls generated by the M:DEVICE (TAB) procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'28'	0————0	DCB address
0	1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

P_1	0————————————————0
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 2

n	t_1	t_2	t_3
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

last word

t_{n-3}	t_{n-2}	t_{n-1}	t_n
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

For this FPT, P_1 must be set to 1.

When the user's program requires tab spacing in the output buffer, this is indicated in the character string by an EBCDIC code of 05. The Monitor responds to such a code by inserting the subsequent character (in the character string) at the column indicated by TAB_i (where TAB_{i-1} was the most recent tab setting used in formatting the current line).

Note that unless the value of $TAB_i < TAB_{i-i}$, data may be lost by being overlapped in the output buffer.

Example:

The procedure call

M:DEVICE M:LO, (TAB, 5, 20, 35)

would result in the following entries in word 15 of the DCB associated with the operator's console:

TAB1 = 5
TAB2 = 20
TAB3 = 35
TAB4 = unchanged

With these tab settings, the EBCDIC (hexadecimal) string

$\underline{05}$C3D6D3E4D4D540F1$\underline{05}$C3D6D3E4D4D540F2

would result in the following typeout:

```
↓             ↓
(col.5)       (col.20)
COLUMN 1      COLUMN 2
```

SKIP TO TOP OF FORM

This call allows the user's program to cause the printer or typewriter associated with a specified DCB to skip to the top of a new physical page. If the printer is already positioned at the top-of-form, no action takes place.

The procedure call is of the form

M:DEVICE [*] dcb name, (PAGE)

where

dcb name specifies the name of the DCB associated with the device that is to be positioned.

PAGE specifies that the device associated with the specified DCB is to skip to the top of the next page.

Calls generated by the M:DEVICE (PAGE) procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'04'	0————0	DCB address
0	1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SET NUMBER OF PRINTABLE LINES

This call allows the user's program to set the number of printable lines per page, for the printer or typewriter associated with a specified DCB.

The procedure call is of the form

M:DEVICE [*] dcb name, (LINES, value)

where

dcb name specifies the name of the DCB associated with the device for which the number of printable lines is to be set.

LINES,value specifies the number of printable lines per page. A maximum of 32,767 lines per page may be specified.

Calls generated by the M:DEVICE (LINES) procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'20'	0————0	DCB address

word 1

P_1	0————————————————0

word 2

*	0————————0	Printable lines per page

P_1 must be equal to 1.

SET LINE SPACING

This call allows the user's program to set the number of spaces between lines printed by a typewriter or printer. It is valid only if the VFC flag in the DCB is set to 0.

The procedure call is of the form

M:DEVICE [*]dcb name,(SPACE,value)

where

dcb name specifies the name of the DCB associated with the device for which the line spacing is to be set.

SPACE,value specifies the number of lines to be spaced after printing a line (a value of either 0 or 1 results in single spacing).

Calls generated by the M:DEVICE (SPACE) procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'25'	0————0	DCB address

word 1

P_1	P_2	0————————————————0

word 2

0————————————————0	Line spacing

P_1 must be equal to 1.

SPECIFY DIRECT FORMATTING

This call allows the user's program to specify whether or not special record formatting is to be done by the Monitor.

The procedure call is of the form

M:DEVICE [*]dcb name, $\left\{\begin{matrix}\text{(DRC)}\\\text{(NODRC)}\end{matrix}\right\}$

where

dcb name specifies the name of the DCB associated with the device for which the special formatting is or is not to be done.

DRC specifies that no special record formatting is to be done for the device associated with the designated DCB (inhibit Monitor formatting).

NODRC specifies that the normal mode of Monitor formatting is to be reinstated for the device associated with the designated DCB.

Calls generated by the M:DEVICE(DRC/NODRC) procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'0B'	0————0	DCB address

word 1

0————————————————0	f_1	0——0

Flag	Significance
f_1	0 means Monitor formatting is not to be inhibited. 1 means Monitor formatting (for card and paper tape devices) is to be inhibited.

SPECIFY VERTICAL FORMAT CONTROL

This call allows the user's program to specify whether or not the Monitor is to interpret the first character of each output image as a vertical format control character.

The procedure call is of the form

M:DEVICE [*]dcb name, $\left\{\begin{matrix}\text{(VFC)}\\ \text{(NOVFC)}\end{matrix}\right\}$

where

dcb name specifies the name of the DCB associated with the typewriter or line printer that is (or is not) to operate under vertical format control.

VFC specifies that the user has inserted a control character in his print image.

NOVFC specifies that the user has not inserted a control character in his print image.

Calls generated by the M:DEVICE(VFC/NOVFC) procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'05'	0————0	DCB address
0	1–7	8–15	16–31

word 1

0——————————————0	f_1	0——0
0–26	27	28–31

Flag	Significance
f_1	0 means no vertical format control is to be performed. 1 means vertical format control is to be performed.

SPECIFY PAGE COUNT

This call allows the user's program to request that the Monitor count output pages, and also to specify in which column this count is to be listed on the output device. The page count will appear at the top of the form, if no header has been specified (see "Specify Output Header", below); otherwise, the page count will appear on the same line as the header. The count will be expressed in decimal form, from 1 to 9999.

The procedure call is of the form

M:DEVICE [*]dcb name,(COUNT,tab)

where

dcb name specifies the name of the DCB associated with the typewriter or printer on which the page count is to be listed.

COUNT,tab specifies the column in which the most significant digit of the page count is to be listed. The value of "tab" must be appropriate for the physical device associated with the DCB.

Calls generated by the M:DEVICE (COUNT) procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'24'	0————0	DCB address
0	1–7	8–15	16–31

word 1

P_1	0——————————————0
0	1–31

word 2

*	0————————0	Leftmost column of page count
0	1–15	16–31

P_1 must be equal to 1.

CHANGE OUTPUT FORM

This call allows the user's program to request a change in the form used on the output device (card punch, typewriter, and line printer). The Monitor informs the operator of the change that is to be made. When the operator has changed the form, he informs the Monitor by an appropriate key-in.

The procedure call is of the form

M:DEVICE [*]dcb name,(FORM, [*]address)

where

dcb name specifies the name of the DCB associated with the device for which the change of form is to be requested.

FORM,[*]address specifies the address of the message (that is to be output to the operator) concerning a change of cards or paper. The first byte of the message must specify the number of bytes in the message.

Calls generated by the M:DEVICE (FORM) procedure have the form

CAL1,1   address

where

address   points to word 0 of the FPT shown below.

word 0

*	X'21'	0————0	DCB address

word 1

P_1	0————————————————0

word 2

*	0————————0	Message address

P_1 must be equal to 1.

CHANGE DEVICE MODE OR RECORD SIZE

This call allows the user's program to change the mode of the device associated with a specified DCB, or to change the logical record size entry (RSZ) in the specified DCB.

The procedure call is of the form

M:DEVICE [*]dcb name,(option)

where

dcb name   specifies the name of the DCB associated with the device for which the change in mode or record size is to be made.

The options are:

BCD   specifies the EBCDIC mode.

BIN   specifies the binary mode.

FBCD   specifies FORTRAN BCD conversion.

PACK   specifies the packed binary mode.

UNPACK   specifies the unpacked binary mode.

SIZE,value   specifies the maximum record size, in bytes.

Calls generated by the procedure have the form

CAL1,1   address

where

address   points to word 0 of the FPT shown below.

word 0

*	X'22'	0————0	DCB address

word 1

P_1	0————————————0	f_3	f_2	f_1	0——0

optional

0————————0	Maximum record size

Flag	Significance
f_1	0 means BCD mode. 1 means binary mode.
f_2	0 means no FBCD. 1 means FBCD.
f_3	0 means unpacked. 1 means packed.

SPECIFY BEGINNING COLUMN

This call allows the user's program to specify that all data output by the card punch (EBCDIC only), typewriter, or line printer associated with a designated DCB is to begin in a specified column.

The procedure call is of the form

M:DEVICE [*]dcb name,(DATA,tab)

where

dcb name   specifies the name of the DCB associated with the output device for which the beginning column is to be specified.

DATA,tab   specifies the column in which the first character of the data output is to appear.

Calls generated by the M:DEVICE (DATA) procedure have the form

CAL1,1   address

where

address   points to word 0 of the FPT shown below

word 0

*	X'23'	0————0	DCB address

word 1

P_1	0————————————————0

word 2

0————————————0	Data tab col.

P_1 must be equal to 1

SPECIFY OUTPUT HEADER

This call allows the user's program to specify an output header (heading) that is to appear at the top of each form.

The procedure call is of the form

M:DEVICE [*] dcb name,(HEADER,tab, [*] address)

where

dcb name specifies the name of the DCB associated with the device on which the header is to appear.

HEADER,tab, [*] address specifies the column number (tab) at which the header is to begin, and the address of the header. The first byte of the header must specify the number of bytes it contains.

Calls generated by the M:DEVICE (HEADER) procedure have the form

CAL1,1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'26'	0————0	DCB address
0	1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

P1	P2	0————0
0	1	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 2

*	0————0	Header address
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 3

*	0————0	Header tab
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

P_1 and P_2 must be equal to 1.

SPECIFY CARD PUNCH SEQUENCING

This call allows the user's program to specify that sequence numbers are to be punched on cards output by the card punch associated with a designated DCB.

The procedure call is of the form

M:DEVICE [*] dcb name,(SEQ[,'id'])

where

dcb name specifies the name of the DCB associated with the card punch that is to output cards with sequence numbers.

SEQ[, 'id'] specifies that sequence numbers are to be punched in columns 77-80 of each card. If a user-defined 'id' is specified, it will be punched in columns 73-76 of each card.

Calls generated by the M:DEVICE (SEQ) procedure have the form

CAL1, 1 address

where

address points to word 0 of the FPT shown below.

word 0

*	X'27'	0————0	DCB address
0	1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

P1	0————0
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

optional

Identification characters
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

NUMBER OF LINES REMAINING

This call allows the user's program to determine the number of printable lines remaining on a page.

The procedure call is of the form

M:DEVICE [*] dcb name (NLINES)

where

dcb name specifies the name of the DCB associated with the device for which the number of lines remaining on a page is to be obtained.

NLINES keyword designating what the procedure call is requesting.

Calls generated have the form

CAL1, 1 address

where

address points to the FPT shown below.

*	X'2A'	0————0	dcb
0	1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Upon return to the caller, SR1 contains 0 if not applicable; otherwise, it contains the number of lines remaining on the current page.

CHECK CORRESPONDENCE OF DCB ASSIGNMENTS

This call allows the user's program a means of determining if two DCBs have been assigned to the same physical device.

The procedure call is of the form

M:DEVICE [*] dcb$_1$ name, (CORRES, dcb$_2$name)

where

dcb$_1$ name specifies the name of a DCB which is to be checked for assignment correspondence with dcb$_2$ name.

CORRES, dcd$_2$ name specifies the name of a DCB which is to be checked for assignment correspondence with dcb$_1$ name.

Calls generated have the form

CAL1, 1 address

where

address points to the FPT shown below

word 0

*	X'2B'	0———0	dcb$_1$
0	1 2 3 4 5 6 7	8 9 10 11 12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

*	0———0	dcb$_2$
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

6. DEBUG PROCEDURES

SNAPSHOT DUMPS

The user's program may employ debug procedure calls to determine when or if a conditional snapshot dump is to occur. The functions performed with such procedure calls are the same as those for the corresponding control commands (see Chapter 2).

Debug procedure calls are of the form

M:mnemonic   'comment', (from[, to])...

where

mnemonic   may be any one of the following: SNAP, SNAPC, IF, AND, OR, COUNT.

'comment'   may be from 1 to 8 alphanumeric characters.

from[, to]   see "SNAP", Chapter 2.

Calls generated by the M:SNAP procedure have the form

CAL1, 3   address

where

address   points to word 0 of the FPT shown below.

word 0

X'00'	0————0
0–7	8–31

word 1

*	0————0	First address to be dumped
0	1–14	15–31

word 2

*	0————0	Last address to be dumped
0	1–14	15–31

word 3

First four characters of comment
0–31

word 4

Last four characters of comment
0–31

word 5

Code for "NOP"
0–31

word 6

Code for "BCR, 0 Z+1" where Z is loc. of CAL1, 3
0–31

Calls generated by the M:SNAPC procedure have the form

CAL1, 3   address

where

address   points to word 0 of the FPT shown below.

Word 0

X'01'	0————0
0–7	8–31

Words 1 through 6 of the FPT have the same form as shown above for the M:SNAP procedure.

word 7

*	0————0	Flag address
0	1–14	15–31

Calls generated by the M:IF procedure have the form

CAL1, 3   address

where

address   points to word 0 of the FPT shown below.

word 0

X'02'	0————0
0–7	8–31

word 1

*	Instruction to load I_1 data into register 0
0	1–31

word 2

*	Instruction to load I_2 data into register 0
0	1–31

word 3

Instruction to branch if specified relation (r) is true
0–31

r	Instruction	
GT	BCS, 1	0
LT	BCS, 2	0
EQ	BCR, 3	0
GE	BCR, 2	0
LE	BCR, 1	0
NE	BCS, 3	0

word 4

0——————————————0

word 5

NOP instruction

word 6

Code for "BCR, 0 Z+1" where Z is loc. of CAL1, 3

word 7

*	0——————0	Flag address

Calls generated by the M:AND procedure have the form

CAL1, 3   address

where

address   points to word 0 of the FPT shown below.

word 0

X'03'	0——————————0

Words 1 through 7 of the FPT have the same form as shown above for the M:IF procedure.

Calls generated by the M:OR procedure have the form

CAL1, 3   address

where

address   points to word 0 of the FPT shown below.

word 0

X'04'	0——————————0

Words 1 through 7 of the FPT have the same form as shown above for the M:IF procedure.

Calls generated by the M:COUNT procedure have the form

CAL1, 3   address

where

address   points to word 0 of the FPT shown below.

word 0

X'05'	0——————————0

word 1

Binary number to start count

word 2

Binary number to end count

word 3

Binary number specifying step intervals

word 4

0——————————————0

word 5

NOP instruction

word 6

Code for "BCR, 0 Z+1" where Z is loc. of CAL1, 3

word 7

*	0——————0	Flag address

MONITOR ERROR CONTROL

M:MERC The Monitor MERC routine enables the user's program to specify an error or abnormal code (see Appendix H) in SR3 and have the Monitor handle it, overriding any user's abnormal or error routines that might otherwise apply.

This call can also be used to return control to the Monitor from a user's error or abnormal routine if that routine can handle only certain codes. If a user's error or abnormal routine is called to handle an abnormal or error condition beyond its capability, it must leave the contents of communication register SR3 intact and call the MERC routine to handle the condition.

The M:MERC procedure call is of the form

M:MERC

Although no parameters are specified in this call, communication register SR3 must contain the error or abnormal code (in byte 0) when MERC is entered. For I/O errors or abnormal conditions, the address of the associated DCB must also be contained in SR3, in bytes 1-3 (right-justified). This information is placed in SR3 by the Monitor, when an error or abnormal condition is detected. When byte 0 of SR3 contains 40_{16} - FF_{16}, the current job is aborted and the message

I/O ERROR xx is logged on OC and LL. It should also be noted that SR1 will contain the address+1 of the offending CAL when the error or abnormal address is entered. It must also be preserved so that MERC can return properly.

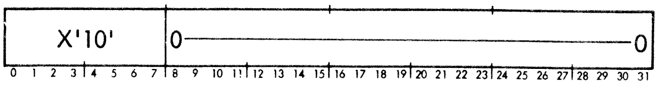
Calls generated by the M:MERC procedure have the form

```
    CAL1,2      address
```

where

address   points to word 0 of the FPT shown below.

word 0



7. USE OF TEMPORARY STORAGE BY LIBRARY ROUTINES

All standard system library routines are entered by a BAL instruction, using current general register 11 as a link register. Arguments are passed to the library routine through current general registers 6 through 9. Current general register 0 contains a pointer to a Task Control Block (TCB) established and maintained by the Monitor. The first two words of a TCB comprise a stack pointer doubleword, and the subsequent words contain additional information used by the Monitor to control the current task.

The library routine can make register contents available for use by pushing them into the temp stack set by the Monitor for the current job. Other kinds of temporary data also can be saved in the temp stack (e.g., trap return information).

One means of storing data in the temp stack is to push the data into the stack, by means of push instructions (PSM and PSW); another is to set aside a block of storage in the stack, by using an MSP instruction, so that data can then be stored in the block by the use of store instructions.

All storage used in the temp stack must be released before the associated routine exits. Storage can be released by a pull (PLM or PLW) or MSP instruction. All registers except those used to return output information must remain unchanged. Note that all push, pull, and MSP instructions must be done indirectly, through current general register 0.

Examples of two different methods of placing data in temporary storage are given below.

```
    R0    EQU      0
    R6    EQU      6
          :
          LCI      4
          PSM,R6   *R0
```

The preceding example causes the contents of current general registers 6, 7, 8, and 9 to be pushed into the temp stack.

```
    R7        EQU       7
    BLOCK1    EQU       100
              .
              .
              .
              LI,R6     BLOCK1
              MSP,R6    *R0
              LW,R7     *R0
```

The preceding example reserves a 100-word block in the temp stack. The address of the top of the block is contained in R7. When the block has been reserved, data can then be stored in it, using the method illustrated below.

```
    R3      EQU       7
    BLW1    EQU       -99
            .
            .
            .
            STW,R3    BLW1,R7
```

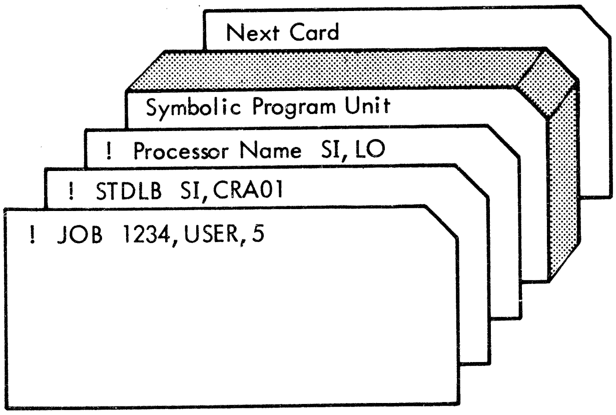
In the example given above, the contents of current general register 3 are stored in the first word of the reserved block.

The area reserved for use by the temp stack is established by a LOAD command TSS specification or is set at 512 words by default.

8. PREPARING THE PROGRAM DECK

The following examples show some of the ways that program decks may be prepared for Monitor operation. Standard system assignments are normally assumed.

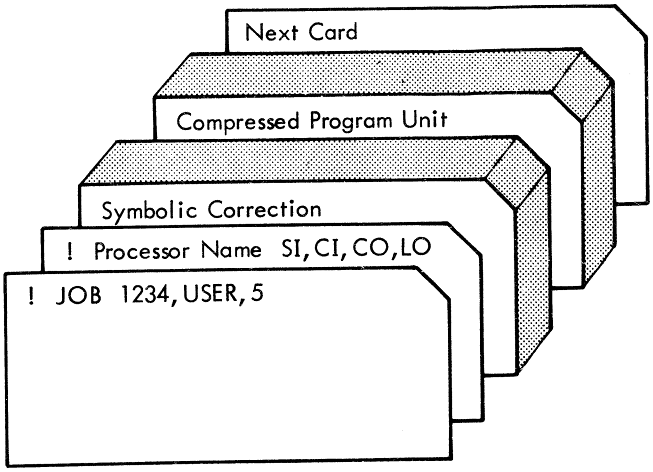
SYMBOLIC DECK TO PROGRAM LISTING



As indicated in the example, a program listing can be obtained even though no binary (object) output is requested. If an explicit assignment for operational label SI had not been specified via a STDLB command, the standard system assignment would have applied by default (as with operational label LO in this example).

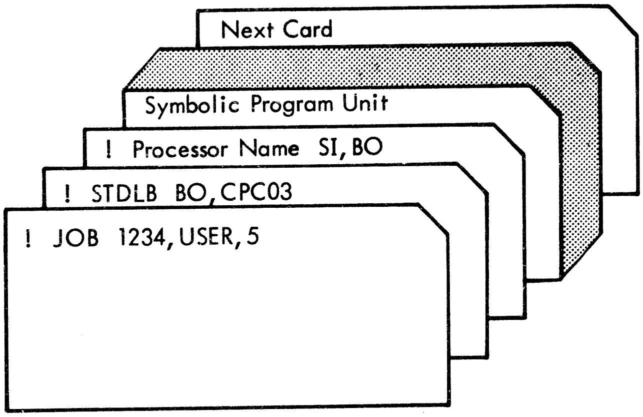
In this and subsequent examples of program decks, the "next card" could be any appropriate control command card such as JOB or FIN.

COMPRESSED DECK UPDATE



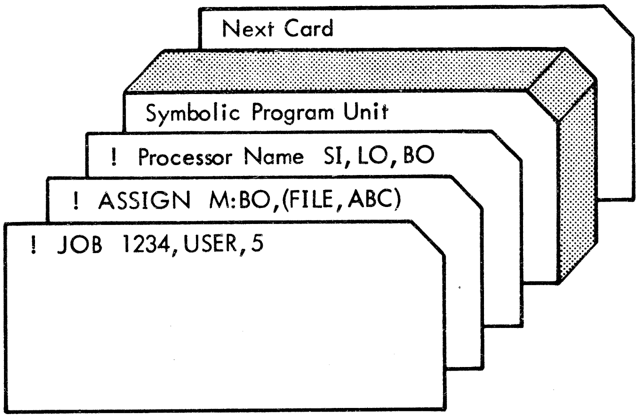
This example shows how a compressed symbolic deck can be updated using a previously output compressed deck and uncompressed corrections as input.

SYMBOLIC DECK TO BINARY DECK



This example shows how a binary deck can be output using an uncompressed symbolic deck as input. A deck produced in this way will be in standard 10 070 object language and may be loaded from a card reader, subsequently, as a binary object module.

SYMBOLIC DECK TO BINARY FILE ON DISC

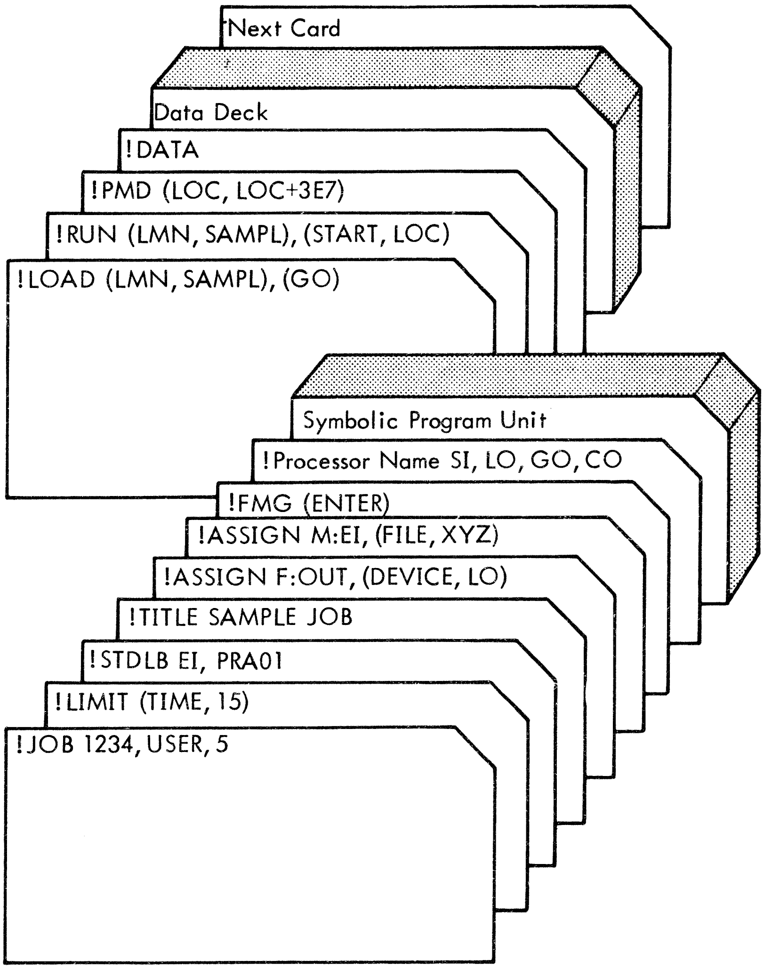


This example shows how a binary (object) file on disc can be formed from the output of a processor. The disc file thus formed may be accessed thereafter by means of the file name ABC. The example specifies that a program listing is to be output also; but this is optional, since only the binary output is retained in disc storage.

The file becomes a permanent user's file. That is, it is placed in the disc area allocated to permanent storage, since all processor output is treated by the Monitor as a SAVE file (see "ASSIGN", Chapter 2).

PROCESS, LOAD, AND EXECUTE

This example shows how a program can be input in symbolic form, then processed, loaded, and executed under Monitor control. If assembly and run-time are in excess of 15 minutes,



the job is aborted and the user logged out. The operational label EI is assigned to a paper tape reader, device number 01 on channel A. The title "SAMPLE JOB" is printed at the top of each page of the assembly listing.

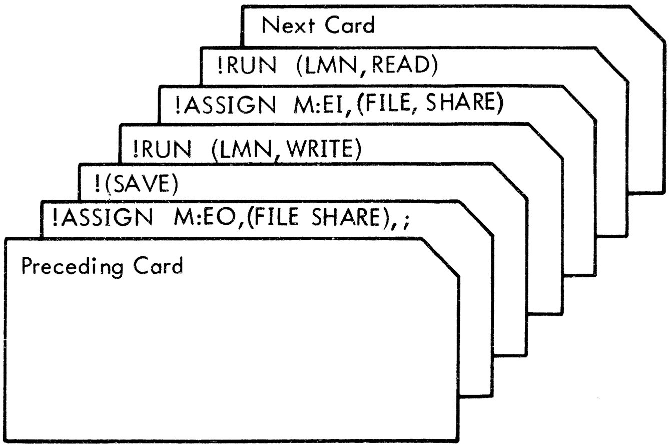
The user's DCB F:OUT is assigned to the LO device (normally a line printer) and file XYZ is input from the EI device (in this example, a paper tape reader).

The processor accepts symbolic input and produces compressed symbolic output as well as binary object code and a program listing on the system devices to which such functions are currently assigned (by standard system assignments).

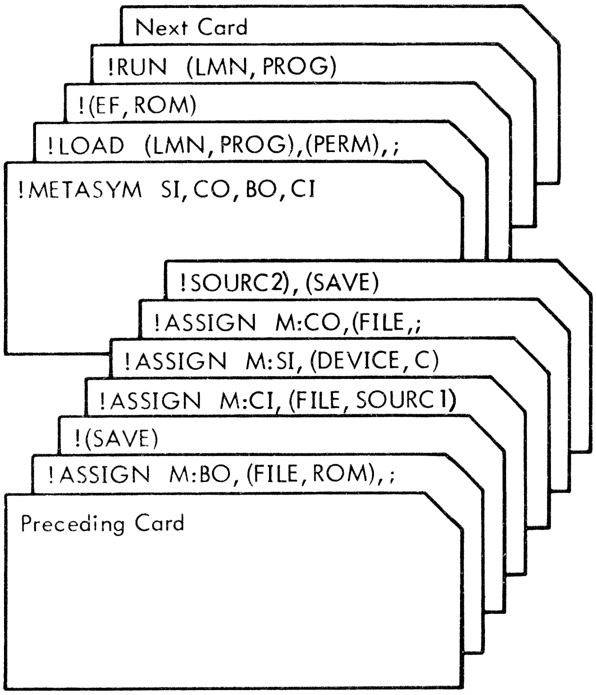
The name "SAMPL" is associated with the load module, and no load map is output. After being loaded into core storage, the program is executed beginning with the instruction at symbolic location LOC. If an error is detected during execution of the user's program, a postmortem dump is taken of the first 1000 words of the program.

CREATE FILE FOR USE BY ANOTHER PROGRAM

This example shows how an output file (SHARE) can be created by one executing program (WRITE) in a user's job and then used by another executing program (READ) in the same job.



UPDATE FILE, OBJECT MODULE, AND LOAD MODULE OF USER'S PROGRAM

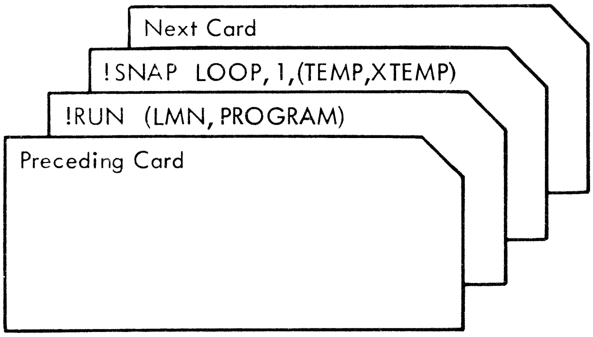


```
Next Card
!RUN  (LMN, PROG)
!(EF, ROM)
!LOAD  (LMN, PROG),(PERM),;
!METASYM  SI, CO, BO, CI
!SOURC2), (SAVE)
!ASSIGN  M:CO,(FILE,;
!ASSIGN  M:SI, (DEVICE, C)
!ASSIGN  M:CI, (FILE, SOURC1)
!(SAVE)
!ASSIGN  M:BO, (FILE, ROM),;
Preceding Card
```

This example shows how a user can accomplish the following:

1. Update a symbolic file (SOURC1 becoming SOURC2) in the user's library.
2. Create a new relocatable object module (ROM) and place it in the user's library.
3. Execute the program so formed.

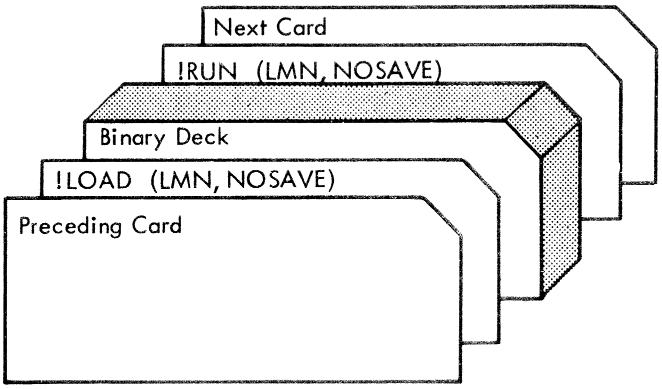
EXECUTE PROGRAM FROM USER'S LIBRARY, USING DEBUG FEATURE



```
Next Card
!SNAP  LOOP, 1,(TEMP,XTEMP)
!RUN  (LMN, PROGRAM)
Preceding Card
```

This example shows how a program (PROGRAM) from the user's private library can be executed, and a snapshot dump of a specified program area (from TEMP to XTEMP) taken.

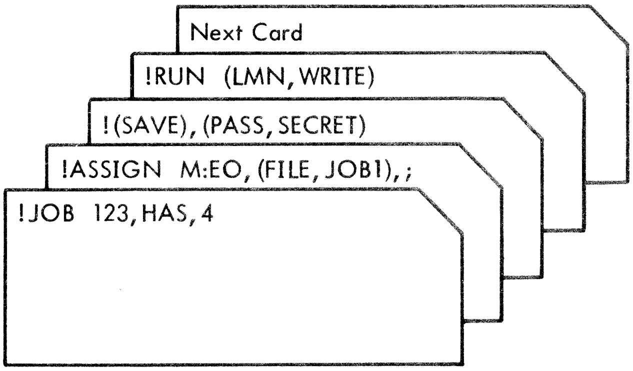
CREATE AND EXECUTE TEMPORARY PROGRAM



```
Next Card
!RUN  (LMN, NOSAVE)
Binary Deck
!LOAD  (LMN, NOSAVE)
Preceding Card
```

This example shows how a user can create a temporary load module (NOSAVE) that can be executed but is released at the end of the job.

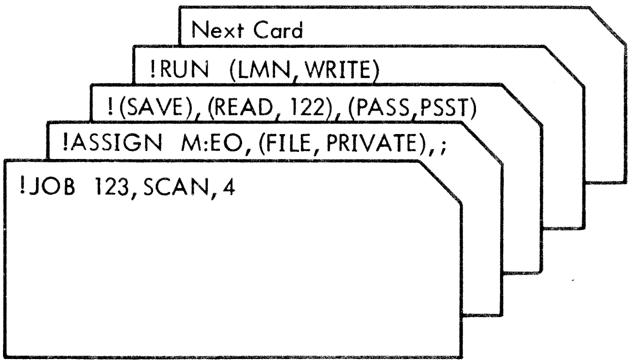
CREATE A PRIVATE FILE



```
Next Card
!RUN  (LMN, WRITE)
!(SAVE), (PASS, SECRET)
!ASSIGN  M:EO, (FILE, JOB1),;
!JOB  123, HAS, 4
```

This example illustrates how a user can create a private file (JOB1) having a password (SECRET). It assumes that a user's program (WRITE) capable of writing into the file (e.g., via an M:WRITE procedure call referencing the M:EO DCB) has been loaded previously.

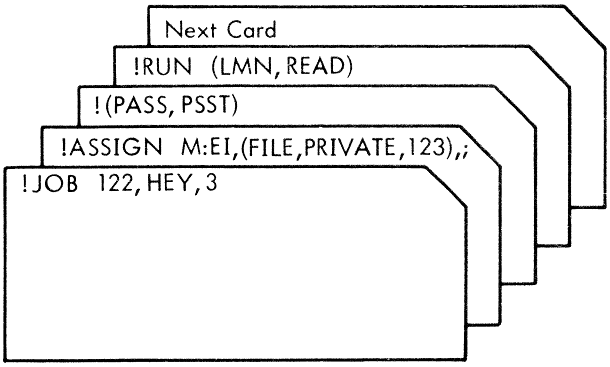
CREATE A PRIVATE FILE HAVING PRIVILEGED READ ACCESS



```
Next Card
!RUN  (LMN, WRITE)
!(SAVE), (READ, 122), (PASS,PSST)
!ASSIGN  M:EO, (FILE, PRIVATE),;
!JOB  123, SCAN, 4
```

Here, a user creates a private file (PRIVATE) having a password (PSST), specifying also that other jobs with account number 122 may read the file (but may not write on it).

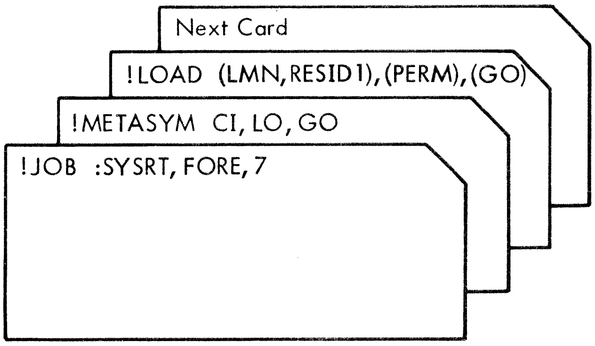
READ A PRIVATE FILE HAVING PRIVILEGED READ ACCESS



```
Next Card
!RUN  (LMN, READ)
!(PASS, PSST)
!ASSIGN  M:EI,(FILE,PRIVATE,123),;
!JOB  122, HEY, 3
```

This illustrates how a user can read a private file (PRIVATE) having a password (PSST) for which privileged read access has been established. Note that the account number associated with the file (123) must be given as well as a password.

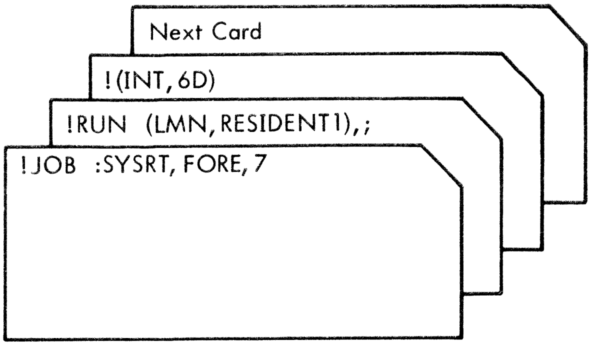
CREATE A RESIDENT FOREGROUND TASK



```
Next Card
!LOAD  (LMN,RESID1),(PERM),(GO)
!METASYM  CI, LO, GO
!JOB  :SYSRT, FORE, 7
```

This example shows how a user can create a resident foreground task (RESIDENT1) by loading from the GO file.

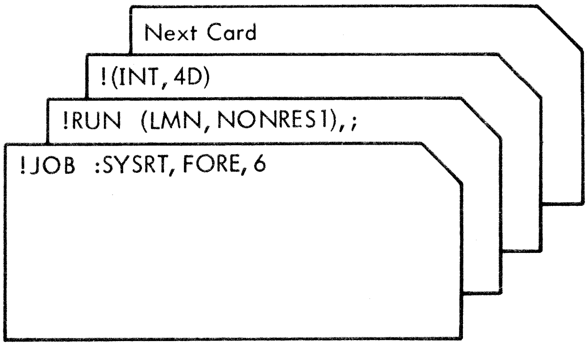
LOAD A RESIDENT FOREGROUND TASK FOR EXECUTION



```
Next Card
!(INT, 6D)
!RUN  (LMN, RESIDENT1),;
!JOB  :SYSRT, FORE, 7
```

Here, a user obtains a foreground task (RESIDENT1) and connects it to an interrupt (6D). The task must have been declared resident at System Generation time or it will not be recognized by the Monitor as a resident foreground task.

CONNECT A NONRESIDENT FOREGROUND TASK FOR EXECUTION



```
Next Card
!(INT, 4D)
!RUN  (LMN, NONRES1),;
!JOB  :SYSRT, FORE, 6
```

This example shows how a user can obtain a nonresident foreground task (NONRES) and connect it to an interrupt (4D). The task must have been declared nonresident at System Generation.

9. REAL-TIME OPERATIONS

This chapter gives a detailed description of how to generate a real-time system, how the foreground tasks are created, loaded, and connected to the interrupts, and what facilities the system provides for the foreground tasks.

The Monitor system provides the capabilities for handling resident, nonresident, and background tasks concurrently. The priority of a foreground task is determined by the priority of its associated interrupt. The lower the interrupt location the higher the priority of the task.

Resident tasks may be connected to any external interrupt, hardware clock, or the clock dedicated to the Monitor.

A common data area called foreground common is allocated in resident memory, if requested at System Generation time (see "System Generation", Chapter 10). This area is accessable to all operating foreground tasks but is protected from background tasks.

Dedication of I/O devices and register blocks to foreground tasks and the ability to connect a resident task or one of its subroutines to the CAL3 or CAL4 trap locations is also provided.

GENERATION OF A REAL-TIME SYSTEM

System Generation provides each installation the means of defining the real-time characteristics of the system. These specifications may be stated by means of control commands. If the control commands fail to provide for specific needs, the installation may supply its own real-time initialization module.

The control command specifications include: memory and disc allocation, dedication of devices, and specification of the attributes of each real-time task.

The following installation control commands (described in Chapter 10) may be used for this purpose:

:DEVICE

:RESERVE

:INTS

:INTR

:TIME

This discussion is confined to those options or specifications that concern the real-time user.

:DEVICE

This command is used to introduce peripheral units and their handlers into the system. It also allows for the dedication of a device to foreground tasks. The management of a device dedicated to foreground usage is entirely under the control of the foreground users.

The foreground user who wishes direct access disc storage may use this command to specify the amount desired. The disc is divided into four areas, as discussed below.

The system area contains the disc bootstrap, initialization routines, and the absolute overlays of the Monitor.

The file area contains all permanent and scratch files. Included in this area are such things as processors, resident and nonresident real-time tasks, user files, user checkpoint storage, etc.

The symbiont area contains all symbiont input and output files.

The backbround checkpoint area contains tasks that have been checkpointed by the Monitor procedure call M:SBACK.

The disc tracks not specifically allocated to the above may be used by foreground tasks. The management of the foreground area is entirely the responsibility of the real-time user.

:RESERVE

This command has several keywords that apply to the generation of a real-time system.

The keywords FIPOOL and FFPOOL specify the number of file index buffers and file blocking buffers to be allocated for use by resident foreground tasks. If specified, these buffers are allocated in the resident Monitor. They must be specified if any resident foreground task uses the Monitor-supplied services for files or labeled tape.

File index buffers are used by the Monitor in locating user files, and for maximum efficiency there should be one allocated for each file opened. File blocking buffers are used by the Monitor to provide for blocking and unblocking of user's records. For maximum efficiency, each open file should have a blocking buffer. Buffer assignment is entirely up to the Monitor. The blocking buffers are 512 words in size, whereas the file index buffers are disc sector size + 12 words. Nonresident foreground tasks are allocated buffers dynamically for Monitor-serviced files and labeled tapes.

The RESDF keyword is used to specify, in decimal, the size of the resident foreground area. This area is allocated starting from the foreground common area down, in multiples of 512. If no foreground common is requested, the resident area is allocated from the high end of core down (see Figure 2). The area not specifically allocated

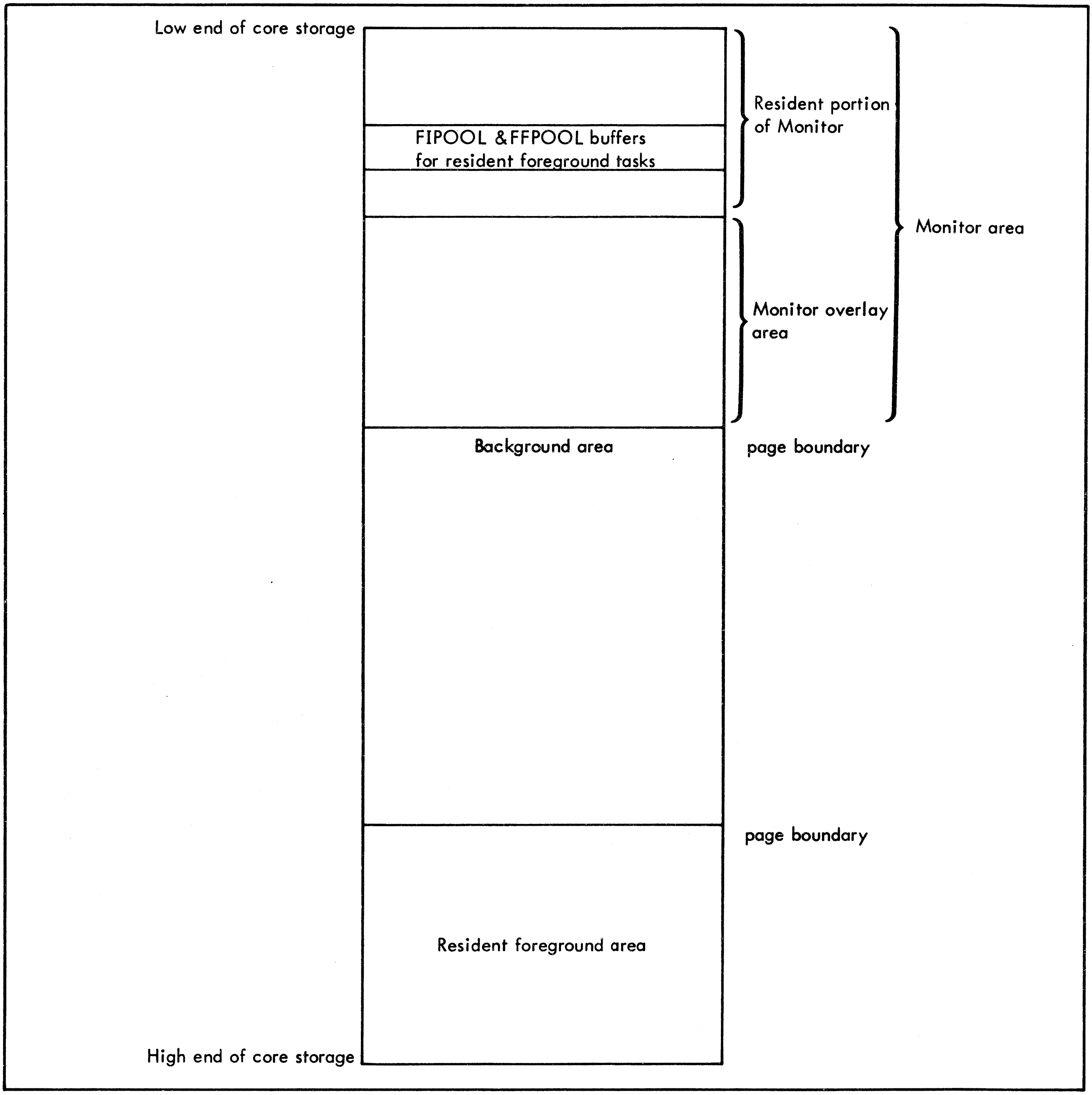


Figure 2. Real-Time System Memory Layout

to the Monitor or to the resident foreground task is available for batch and nonresident foreground operations. Nonresident foreground tasks operate in the same memory area as batch jobs.

This means that an occurrence of a nonresident interrupt will cause the batch area to be checkpointed. If a nonresident task is interrupted by another nonresident task of higher priority, the lower-priority task will be checkpointed and the higher-priority task loaded. Upon completion of the higher-priority task, the environment for the lower-priority task will be reinstated and control will be returned to the point of interruption.

:INTS

This control command allows interrupts to be connected to tasks that are incorporated as part of the Monitor. This is accomplished by specifying the interrupt location to which the task is to be connected and the entry point in the task. An entry point may be an external label or an absolute location. If it is an external label, System Generation will generate the necessary external reference to that label. This reference will be satisfied by including in the Monitor a relocatable binary module with the appropriate external definition.

The DIRECT option specifies that the interrupt is to be connected directly to the task; i.e., the Monitor's interrupt service routine is not entered when the interrupt occurs, but execution of the XPSD instruction in the interrupt location gives control to the task at its entry point. If DIRECT is not specified, the task is entered via the Monitor's interrupt service routine which saves all of the registers and other pertinent information.

The rblock option allows for the specification of a dedicated register block. The management of any register blocks so specified are the responsibility of the real-time user. No checks are made to guarantee that a specified register block exists or that it is not used by several tasks. In fact, several tasks may use the same dedicated register block (but different registers).

:INTR

This control command is used to specify the load module names of all foreground tasks, whether resident or nonresident, and the interrupt locations to be associated with each load module. The priority of each task is determined by the priority of its associated interrupt location.

Each foreground load module is formed and connected under Monitor control, as a batch job, by running with the account number :SYSRT.

The "number" :SYSRT is the only account number recognized for real-time operations. The Monitor LOAD control command is used to construct the load modules and the RUN control command is used to connect a load module to an associated interrupt.

The same load module name may be associated with several different interrupt locations. When the task is loaded for execution via the RUN control command, the specified interrupt is connected to the transfer address associated with the load module, or the START address in the RUN command, if specified. The other interrupts associated with the task may be connected by the task itself with the M:ARM or M:DISARM calls to the Monitor.

The RESD keyword specifies that, when the named load module is loaded for execution (via a RUN command), it will be loaded into the resident foreground area. The keyword MASTER may be specified for a task which is declared resident (RESD) or nonresident (NRESD). If a task is to be loaded into the resident foreground area, and if the associated interrupt is to be directly connected to the task, MASTER must be specified. Tasks entered by the Monitor's interrupt service routine are entered in the master mode if MASTER is specified.

The keyword NRESD specifies that the named task is to be nonresident. If NRESD is specified, the task may not be entered directly.

A program that operates in the master mode may execute privileged instructions; whereas, a task operating in the slave mode may not execute any privileged instructions.

:TIME

This control command may be used to specify resident foreground tasks that are to be connected to the Monitor's clock (location $5B_{16}$) via the Monitor's clock interrupt service routine.

This command must be used to name those tasks that are to be connected to the Monitor's clock. This special class of tasks is treated as an extension to the Monitor. The time interval in which they are to be entered is specified in the RUN control command.

Tasks connected to the Monitor's clock are entered in the master mode with current general register 11 containing the return address to the Monitor. Registers 0-4, 12-15 may be considered volatile by the foreground task.

INITIALIZATION MODULES

When structuring the Monitor (see Chapter 10), modules may be included that will provide installation initializing functions. These modules will be executed only when bootstrap loading the system.

The Monitor initialization procedures exist in two separate segments, as discussed below.

The first segment is responsible for initializing disc from tape and only ocurs when bootstrapping from tape. Any module included in the CLS segment of the Monitor between files CLS and CLS2 on the tree, with an externally defined entry point of USNRINIT1 or USNRINIT2, will be included in this phase and will be entered after the system has loaded the disc. The system will transfer control to the entry point in the master mode with general register 11 containing the return address. If both entry points are specified, USNRINIT2 will be entered before USNRINIT1.

The second segment is basically responsible for initializing the interrupts, CALs, and traps. This phase occurs on either a tape or disc bootstrap after all other initialization has been completed. and before the system enters the "wait" state. Any module included in the TYPR segment of the Monitor with an externally defined entry point of USRINIT1 or USRINIT2 will be included in this phase. Control will be transferred to the entry point in the master mode with general register 11 containing the return address. If both entry points are specified, USRINIT2 will be entered before USRINIT1. These programs may not call on the Monitor via the normal procedure calls.

CREATION OF FOREGROUND LOAD MODULES (TASKS)

Foreground load modules are formed under Monitor control by running a job under the account number :SYSRT. As stated previously, :SYSRT is the account number given to real-time operations.

The Monitor control command "LOAD" is used to direct the loader to form a relocatable load module from relocatable object modules. The resulting program may be entered into the real-time account file.

The installation control command ":RESERVE" allows for the specification of a resident foreground pool. If the foreground load module is to use this pool, the FCOM option must be specified on the LOAD control command. The keyword "FCOM" instructs the loader to connect all program references to F4:COM to the resident foreground common pool. Both resident and nonresident foreground tasks may share this pool. The management of this pool is entirely the responsibility of the foreground users.

The LOAD control command may be used at any time to replace previously constructed foreground load modules.

TASK INITIATION

The RUN control command is used to connect a load module to its associated interrupt. The interrupt specified in the RUN command is connected to the transfer address associated with the load module, or to the START address specified in the RUN control command. Other interrupts associated with the task may be connected by the task itself with the M:ARM or M:DISARM procedure calls. A RUN control command also may be used to replace a task that is "locked" (see "Operator Control of Foreground Tasks").

The name of the load module (LMN,name) must correspond to the name specified in either the INTR or TIME installation control commands. The attributes of the load module are specified at the time the system is generated. Any options specified in the RUN control command must agree with the System Generation specifications or the RUN control command will not be honored.

The INT option specifies the interrupt location to be associated with the real-time program. If the named load module has been designated as a resident foreground task, it will be loaded into the resident foreground memory based on the value specified in the BIAS option. If no BIAS is specified, the resident foreground task will be loaded at the bias specified when it was created. The management of this resident foreground area is the responsibility of the real-time user, and it is recommended that user obtain a load map in order to maintain proper management of this area.

If the named load module has been designated as a nonresident foreground task, it will be loaded when the respective interrupt occurs.

The keyword DISABLE, which may be specified on the INT option, states that the interrupt is to be armed and disabled; otherwise, it will be armed and enabled. If INT is specified, neither TIME nor CLOCK may be specified in the RUN command.

The TIME option specifies that the specified task is to be connected to the Monitor's clock (location X'5B'). It will be entered via the Monitor's clock interrupt service routine (cyclically) in the time interval specified. Tasks connected to the Monitor's clock are entered in the master mode with the current general register 11 containing the address of the return to the Monitor. General registers 0-4, 12-15 may be considered volatile by the designated task.

If TIME is specified, the INT, CLOCK or DIRECT options may not be specified.

The CLOCK option is used to connect a foreground task to a specified clock. The "value" specifies the time interval, in units of the clock's resolution, to which the clock is to be set. The keyword DISABLE, which may be specified in the CLOCK option, states that the designated clock is to be armed and disabled. If not specified, the designated clock will be armed and enabled. Tasks connected to a clock are always connected directly.

If CLOCK is specified, neither TIME nor INT may be specified in the RUN command.

The DIRECT option specifies that the foreground task is to be entered directly each time the associated interrupt or clock becomes active. The MASTER and RESD options must have been declared as attributes of the task at System Generation time in order for the option to be valid. The task will be entered in the master mode and is responsible for saving and restoring any machine environment it changes. It is also responsible for returning control to the point at which the interrupt occurred.

This option also allows for the specification of a register block. If a register block is specified, the register block will be enabled upon entry to the task.

If DIRECT is not specified, the task will be entered through the Monitor's interrupt service routine and the Monitor will automatically save and restore the machine environment, including the interrupt location.

OPERATOR CONTROL OF FOREGROUND TASKS

The primary function of the unsolicited key-ins is to aid in controlling foreground tasks. The operator may command the system to change the status of a known task by means of these key-ins. A known task is one that has been created by means of LOAD control command and initiated by means of a RUN control command.

The form of the unsolicited key-ins is

```
!F name, m
```

where

name specifies the task name

m specifies the change of state or action desired.

The values of "m" recognized by the Monitor are shown in the following example.

m	Action desired
X (abort)	Causes the specified task to be aborted (forces control to be relinquished to the Monitor), if currently active. If the task is not active, the X key-in disarms the interrupts upon their next occurrence. The result of this key-in on an active task is the same as though the task had executed an M:XXX procedure call. The task may be armed by an I key-in or an M:ARM procedure call.
S (suspend)	Causes the specified task to be suspended. If the task is currently active, its associated interrupts will be disarmed when the task relinquishes control to the Monitor. If the task is not currently active, the interrupts associated with the task will be disarmed upon their next occurrence. If the task is resident, its associated DCBs will not be closed on exit. The task is marked as being suspended and may be rearmed by the C key-in.
L (lock)	Causes the specified task's interrupts to be disarmed and prohibits the task from being initiated by an I key-in or an M:ARM procedure call. If the task is currently active, it will be disarmed when the task relinquishes control to the Monitor. If the task is not active, its associated interrupts will be disarmed upon their occurrence. A locked task may be initiated only by !RUN control command. A locked task may be unlocked by a C key-in. An unlocked task may be armed by an I key-in or an M:ARM procedure call.
C (continue)	Causes the interrupts of the associated task to be armed and enabled, if the task has been suspended. If the task is locked, the lock condition is cleared (unlocked) and the task may then be initiated either internally or externally.
I (initiate)	Causes the highest priority interrupt associated with the specified task to be armed and enabled. The primary function of this key-in is to enable tasks that have been armed and disabled by the RUN control command or disarmed by means of the M:DISARM, M:TERM, or M:XXX, procedure calls or the X key-in. It may also be used for initiating an unlocked task.

For foreground tasks that are connected to the Monitor's clock via the :TIME installation control command, the key-ins work differently. Only the S and I key-ins apply. The I key-in causes the Monitor to give control to the task (i.e., initiates the task) at the time interval specified and the S key-in causes the Monitor to discontinue the task (i.e., suspends the task).

SYSTEM PROCEDURES FOR REAL-TIME FOREGROUND TASKS[t]

The following calls are available for foreground tasks only. If a background program makes the call it will be aborted.

M:TRIGGER

If the specified interrupt is armed, the interrupt is then triggered by a WD instruction, otherwise the trigger function is not performed and the call is treated as an NOP.

M:DISABLE

If the specified interrupt is currently armed and enabled, it is disabled via the WD instruction, otherwise the call is treated as an NOP.

M:ENABLE

If the specified interrupt is armed and disabled, it is enabled, otherwise the call is treated as an NOP.

M:DISARM

If the disarm call is also a connect function, the interrupt is disarmed and then the connect function is performed. If the disarm call does not involve a connect function and the interrupt is currently armed, a flag is set in the interrupt table entry for the interrupt location. The interrupt will then be disarmed the next time the interrupt level is cleared after becoming active. If the interrupt is already disarmed and a connect is not specified, the call is treated as an NOP.

M:ARM

If the connect option is specified on the call, the interrupt is first disarmed, then connected, and then armed. If the connect function is not specified, the interrupt is armed if disarmed. Otherwise, the call is treated as an NOP.

M:CAL

The specified CAL trap location is connected to the specified address. The PSD used by the XPSD instruction in the trap location has the interrupt inhibits set.

M:DCAL

The specified CAL trap location is connected to the Monitor's abort routine. If a task does a CAL for the specified trap location, the task will be aborted.

M:SLAVE

The master-slave mode bit in the PSD is set to indicate the slave mode.

[t] These procedures are discussed in detail in Chapter 4 of this manual.

M:MASTER

The master-slave mode bit in the PSD is set to indicate the master mode if the MASTER option was specified for the task at system generation time. Otherwise, the task is aborted.

M:SBACK

The entire background area is written to the foreground area on the disc and the write locks are set to an access code of 10. The address of the first location in the background area is returned in SR2 and the size, in pages, is returned in SR1. A task may save the background area only once. If the task does not restore the area with a M:RBACK call, the area will be restored by the Monitor when the task exits.

M:RBACK

Restores the background area saved by the M:SBACK call. If the task did not save the background area, the task will be aborted.

M:TERM

The message "!!TASK name TERMINATED" is output on the OC device and all of the interrupts associated with the task are disarmed. A terminated task may be reinitiated by the I key-in or the M:ARM procedure call.

M:SXC

Causes control to be passed to the specified address for all exits from the task, whether normal or abnormal. The type of return is communicated in SR1. The type of returns are as follows:

exit	code
normal-M:EXIT	X'00'
abort-M:XXX	X'40'
error	X'80'
termination-M:TERM	X'20'
I/O error abort	X'02'
trap error abort	X'01'

An unsolicited abort (X) key-in will abort the task without passing control to the user's program even if it has requested exit control.

M:RXC

This call resets the user's exit control so that all exits are processed by the Monitor.

FOREGROUND RESTRICTIONS

MONITOR CALLS

If the following procedure calls are made by resident foreground programs, the current call will override any previously made by resident foreground programs.

M:STIMER
M:STRAP
M:TRAP
M:TTIMER
M:TRTN
M:INT

The following procedure calls will be ignored if made by resident foreground programs:

M:CHKPT
M:RESTART
M:LDTRC
M:LINK
M:GL
M:GCP
M:FCP
M:GP
M:FP
M:SMPRT

Any resident task connected to a clock (Monitor's clock included) may not make any calls on the Monitor for service or perform any I/O, because the clock interrupts have a higher priority than the I/O interrupt. Furthermore, a resident task connected to a clock must be connected directly. For this reason, nonresident tasks may not be connected to clock interrupts, since I/O is involved in loading the tasks at the time of interrupt occurrence.

The following procedure calls will be ignored if made by nonresident foreground programs:

M:CHKPT
M:RESTART

FOREGROUND FILES

A real-time foreground task may not create new files, but may open a file in any mode; similarly, it may not release files. Foreground tasks are restricted to updating or reading files from their own account and reading files from other accounts. A batch job running under a foreground account may prevent a foreground job from updating a file. If this occurs, an abnormal code of 14 will be passed at the time the foreground file is accessed.

A batch job running under a foreground account (foreground batch) may add files to the foreground file directory, but in so doing may prevent foreground jobs from obtaining access to the directory. If this occurs, an abnormal code of 20 will be passed to the foreground task when the foreground file is accessed. Care should be exercised in running foreground batch jobs under foreground accounts. As a general rule, a batch job should not be run under the foreground account for any reason (e.g., tó create a file, initiate a foreground task, etc.) if the file directory for the real-time account is in danger of being used by a foreground task. This is because running a background job causes temporary files to be created under the job account, which may prevent foreground tasks from accessing the directory.

SAVING ENVIRONMENT AT INTERRUPT TIME

The interrupt service routine for resident tasks that are centrally connected does the following:

1. Pushes all of the registers into the Monitor's temp stack.
2. Saves the contents of CJOB (location X'4F) and replaces the contents with the address of RJIT (Job

Information Table for resident tasks) in the address portion and the priority of the interrupt in byte zero.

3. Saves the run-status, I/O error code, I/O error and abnormal addresses, TCB address, and the link address to the task's portion of the DCB name table.

4. If the task is an overlay task, the address of its tree table is stored in the tree table address entry in RJIT.

5. Sets the run-status in RJIT to zero and stores the TCB address and DCB name table link address in their respective entries in RJIT.

6. Saves the Monitor overlay status (i. e., the numbers of the Monitor overlay segments that are currently resident).

Saving the environment before initiating nonresident tasks is different, since the interrupts for nonresident tasks are queued. The Monitor checks the queue at certain times, and if a task is waiting, it is initiated when the Monitor checks the queue. All of the registers have already been saved and the saving of the Monitor's overlay status is not necessary. The contents of CJOB are saved and replaced with the address of NRJIT (Job Information Table for nonresident tasks) in the address portion and the priority of the task in byte zero. Then NRJIT and the background memory area are checkpointed to disc, the task is loaded into the background area, and the necessary information is stored in NRJIT.

10. SYSTEM GENERATION

System Generation provides a means of forming a Monitor system adapted to the specific requirements of the user's installation. This is done by processing a master system tape and a set of installation control commands. Mnemonics for installation control commands are not preceded by an exclamation character, but begin with a colon (:) instead. Any continuation records for installation control commands must also begin with a colon in the first column.

Monitor System Generation comprises two passes, the second of which may be omitted. The function of the first pass is to accept a master tape and update elements, select desired elements from each, and generate a new master tape. The second pass accepts the selected elements and forms a complete Monitor system. The resulting system is output to magnetic tape, so that the Monitor system can be loaded at any time.

The various processes comprising System Generation must be performed under the control of a Monitor system having at least minimal BPM capabilities. That is, in order to generate a Monitor one must already have a Monitor in the machine. If there is no Monitor in the machine it is necessary to load one, using the standard Monitor bootstrap procedure (see Chapter 11), before attempting System Generation. It should be clearly understood that generating a Monitor and loading a Monitor are separate processes and the performance of either does not, in itself, necessitate the performance of the other.

SUMMARY OF SYSTEM GENERATION

System Generation is run as a batch job as follows:

- The system account file must contain load modules PASS1, PASS2, and DEF.

- The System Generation procedure must run under a different account number then the system account. This is to preserve the integrity of the current operating system while System Generation is in process. Also, M:BI, M:BO, and M:PO must be assigned. Because System Generation is a lengthy process, TIME and LO limits should be set at 60 minutes and 500 pages, respectively.

The PASS1 processor accepts elements from the master system tape (M:BI) and update modules from M:EI. The M:EI DCB is normally assigned to the card reader. A new master tape is output through the M:BO DCB. This new master tape contains the modified bootable System Generation as defined in the bootstrap procedure. The desired object modules will remain on disc. The assignment of M:BI should only include input serial number(s). The assignment of M:BO should specify DEVICE, MT and output serial number(s). The following control commands are recognized by PASS1.

```
:LABEL
:SELECT
:UPDATE
:SYSWRT
!EOD
```

PASS1 accepts a master tape and update elements as input, places the updated master elements on disc to be used as a data base for subsequent processes, and (optionally) creates a new master tape. When entered, the PASS1 processor reads and processes control cards until either an !EOD or a :SYSWRT card (see ":SYSWRT", below) is encountered. It performs a tape-to-disc transfer of all of the files named in the :SELECT commands (see ":SELECT", below) read from the BI device and also named in a :UPDATE command (see ":UPDATE", below). The :LABEL cards (see ":LABEL", below) identify the various update decks. PASS1 then terminates if an !EOD card was encountered. PASS1 writes a new master tape, using the M:BO DCB if :SYSWRT was encountered. This new master tape contains the following elements.

1. A bootstrap loader.

2. The load module M:MON from the :SYS (system) account, written in core image segments (i.e., capable of being booted into the "target machine").

3. A tape label.

4. Eight labeled files comprising seven load modules and one null file. The seven load modules are M:MON, PASS1, PASS2, DEF, LOADER, CCI, and FMGE (all from the :SYS account). If any of these are missing, a diagnostic is generated. The null file is named "LASTLM".

5. All of the sequential disc files of the current job account, each disc file becoming a labeled tape file.

A PASS2 control command will terminate the selection and update phase of System Generation. If an !EOD card is used in place of a :SYSWRT card, no BO tape is produced.

The PASS2 processor is responsible for defining the Monitor system for a specific installation. Standard modules will be replaced by specific installation requirements. In addition to defining the Monitor, the desired installation library may be formulated.

The control commands recognized by the PASS2 processor are:

```
:STDLB       :DLIMIT
:DEVICE      :ABS
:SDEVICE     :INTS
:MONITOR     :INTR
:RESERVE     :TIME
```

The control commands, if specified, must be in the above order.

The PASS2 processor acts similarly to a compiler, in that it reads source statements or control cards prepared by the programmer and generates output to the disc. The output is in load module form and is intended for inclusion in a Monitor (M:MON) that will be formed later. The Monitor elements formed by PASS2 are installation-dependent. PASS2 commands and the modules (files) that they cause to be formed are listed below.

Command	File or Files Formed

:STDLB }
:DEVICE }
	IOTABLE

:SDEVICE	M:SDEV
:MONITOR	M:CPU and M:JIT
:RESERVE	M:FCOM, M:RESDF, M:RJIT, and M:NRJIT
:DELIMIT	M:DELIMIT
:ABS	M:ABS

:INTS }
:INTR }
	M:INT, M:INTLOC, M:RJIT, and M:NRJIT

:TIME	M:TIME

The next step in System Generation is the formation of load modules on disc. A minimum set of such modules would include M:MON (the target machine Monitor itself), LOADER, CCI, and FMGE. To this must be added any desired optional processors and the system library. This might include such modules as PASS1, PASS2, DEF, PFIL, SYMBOL, FORTRAN or FORTRANH, COBOL, METASYMBOL, OLAY, etc. Normally, the ROMs comprising these processors will be available on disc from PASS1.

An OVERLAY, DEF, or ASSIGN control command will terminate the definition of the installation modules. The OVERLAY processor operates under normal system rules and is responsible for creating the desired Monitor and processor overlays.

After the initiation of PASS2 but before the DEF command, load modules for the two versions of the Monitor's resident loader must be created. These two versions should be identical except that the unsegmented version (for LOAD and OVERLAY commands) must be loaded without a TREE command and must have the load module name "LOADER", and the segmented version (for OLAY commands) must be created with a TREE command included as part of the control command sequence and must have the load module name "OLAY".

The DEF processor is responsible for the formation of a bootable system tape with all load modules that were under the account number under which !DEF was run. When booted from tape, the system will load itself, all load modules, and a disc bootstrap on disc. The disc bootstrap will maintain the current files, whereas the tape bootstrap will initialize the disc.

DEF writes a master tape (using the M:PO DCB) which contains the following elements.

1. A bootstrap loader.
2. The M:MON load module from the current job account (as core image segments capable of being booted into the target machine).
3. A tape label.
4. All disc files specifically named on the DEF card (which may include object modules).
5. All keyed disc files that are not synonymous (e.g., load modules).
6. All synonymous keyed disc files (e.g., library files).
7. A null file named "LASTLM".

The structure of the various Monitor system tapes is as follows:

1. BI and BO tapes:

Record	Contents
1	Bootstrap Record
2	Root of Monitor
3	Tree of Monitor
4	Initialization
5-34	Segments of Monitor

This is followed by a tape label and the files M:MON, PASS1, PASS2, DEF, LOADER, CCI, FMGE, and LASTLM.

Following these load modules are all of the relocatable object modules selected or updated during the PASS1 procedure that generated the tape.

2. PO tape:

   The structure of the tape up to the tape label is identical to that of BI and BO. Following that are files for all of the load modules formed during PASS2.

The operation of the first pass of System Generation is shown graphically below in Figure 3.

The peripheral devices indicated in Figure 3 must be defined for use by System Generation. Input from the operator's console is assumed by the program, but all other devices must be defined before any other action takes place. Thus, the first installation control command should be a :GENCHN command.

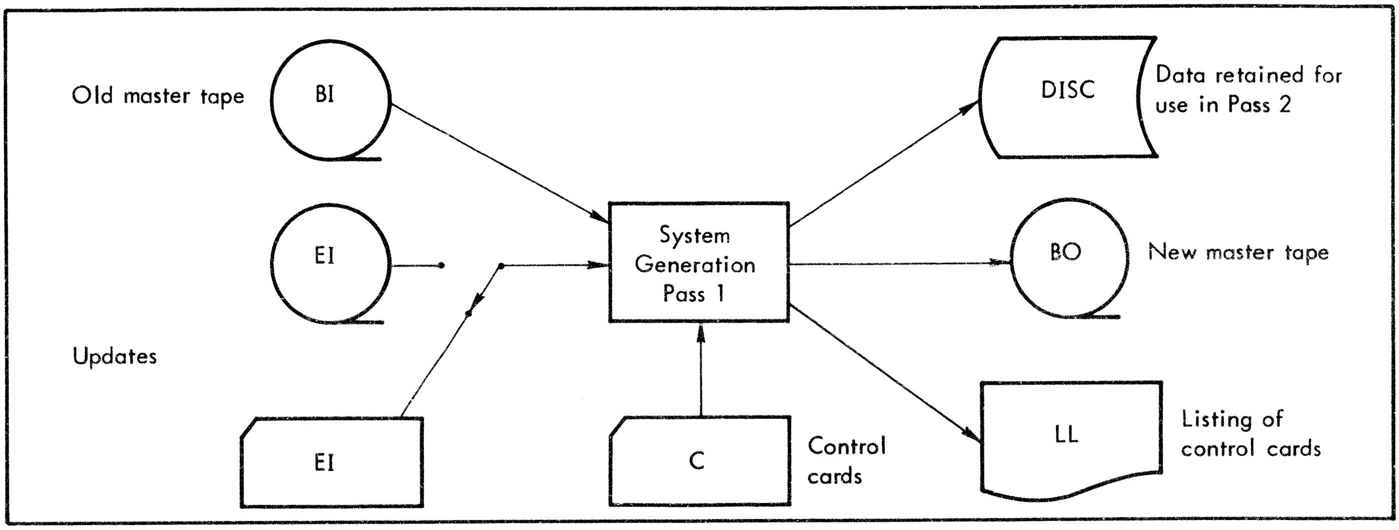


Figure 3. Pass 1 of System Generation

!PASS1 The !PASS1 control command instructs the system to enter the first-pass mode of processing.

The !PASS1 control command has the form

```
!PASS1
```

When this command is encountered, no further commands beginning with ":GEN" will be accepted.

Installation control commands that may be used during pass 1 of System Generation are :SELECT, :UPDATE, :LABEL, and :SYSWRT.

:SELECT The :SELECT installation control command may be used to select (for inclusion in the new master system tape) any of the standard Batch Monitor configurations and any of the standard libraries and processors from the BI device. It may also be used to select any of the relocatable elements comprising such standard systems and programs.

The :SELECT installation control command has the form

```
:SELECT (option)[,(option)] . . . [,(option)]
```

where the options are

FILE, file name ... specifies the name by which an element of a standard processor, library, or Monitor configuration is identified. The first two characters of the file name identify the system type (i.e., "A:" for assemblers, "F:" for FORTRAN compilers, "L:" for libraries, and "M:" for Monitors).

:UPDATE The :UPDATE installation control command may be used to update the desired master system by selecting elements and/or systems from the EI device. Each selected element or system will be added to the master system, replacing any previously selected element or system having the same name. If EI is from the card reader, each element must be preceded by a :LABEL name card (where "name" is the name of the element) and followed by an !EOD card.

The :UPDATE installation control command has the form

```
:UPDATE (option)[,(option)] . . . [,(option)]
```

where the applicable options are the same as those for the :SELECT command (see above).

:LABEL The :LABEL installation control command may be used in a BI or EI card input sequence (see ":SELECT" and ":UPDATE", above) to specify the file name of the data deck following the :LABEL command. The card input deck sequence must be arranged as shown in Figure 4.

The :LABEL installation control command has the form

```
:LABEL, name
```

where

name specifies the file name (up to 8 alphanumeric characters) of the following data deck.

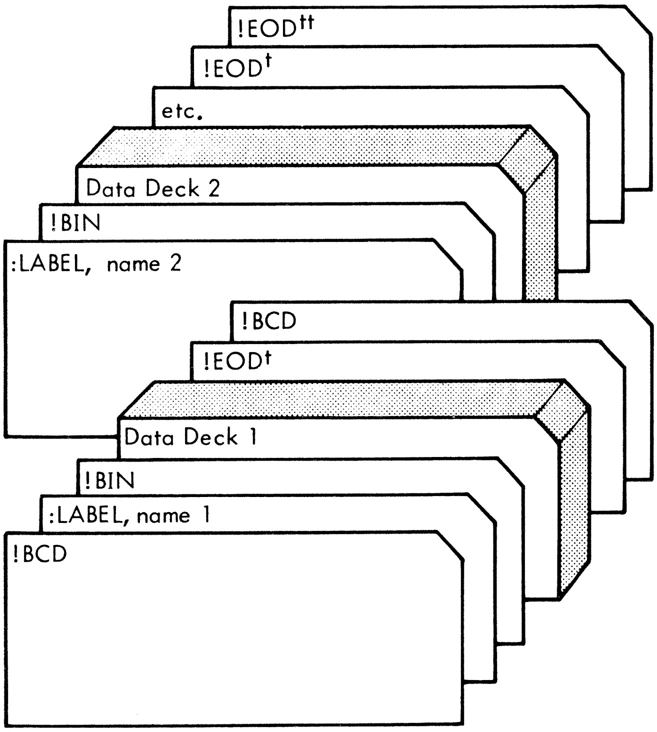


Figure 4. Card Input Sequence for Selecting or Updating Monitor Files

:SYSWRT The :SYSWRT installation control command may be used to cause all elements previously selected by :SELECT or :UPDATE commands to be output on the BO device. If the :SYSWRT command is not used, the object modules comprising the master system elements are retained in disc storage but no master system tape is output.

The :SYSWRT installation control command has the form

```
:SYSWRT
```

!PASS2 The !PASS2 control command instructs the system to enter the second-pass mode of processing.

The !PASS2 control command has the form

```
!PASS2
```

Installation control commands that may be used during the second pass of System Generation are :STDLB, :DEVICE, :SDEVICE, :MONITOR, :RESERVE, :DLIMIT, :INTS, :ABS, :INTR, :TIME, and :DLIB. The form and functions of these commands are described below.

:STDLB All standard Monitor operational labels must be defined by means of :STDLB installation control commands. :STDLB commands may also be used to define an optional set of operational labels for foreground logical devices.

The :STDLB installation control command has the form

```
:STDLB (label, name[, FORE]) [,(...)...]
```

where

label specifies a Monitor operational label or a foreground operational label. All Monitor or foreground operational labels must consist of one or two alphanumeric characters.

name specifies a physical device name to which the operational label is to be assigned (see Tables 2, 3, and 4, in Chapter 3).

and the options are

FORE specifies that the device may be used by foreground tasks only. If FORE is omitted, the device may be used by foreground and background tasks.

If an operational label is to be assigned to a device to which another operational label has already been assigned, the previously assigned operational label may be substituted for the physical device name and no additional parameters need be specified. For example, if the operational label LO has been assigned to a line printer by the installation control command

```
:STDLB(LO, LPA15)
```

then the operational label DO could be assigned to the same device by the command

```
:STDLB (DO, LO)
```

and, if LO was designated as FORE only, DO is similarly designated.

If no :STDLB command is encountered, the following defaults apply.

C = LI = SI = BI = CI = EI = CRA03

OC = TYA01

LO = LL = DO = SL = LPA02

PO = BO = SO = CO = AL = EO = CPA04

:DEVICE The :DEVICE installation control command is used to introduce peripheral units and their handlers into the Monitor system.

[t] End of data

[tt] End of file

The :DEVICE installation control command has the form

```
:DEVICE name[,number] [,(option)] [,(option)]...
```

where

name specifies the device name (see Tables 2, 3, and 4 in Chapter 2).

number specifies the number of the disc (if two discs have the same device name). The disc number may be either 0 or 1.

and the options are as follows:

INPUT/OUTPUT/IO specifies whether the device is to be used for input, output, or both. If this option is omitted, IO is assumed.

HANDLER, name specifies the name of the I/O handler to be used with the device. If this option is omitted, the standard handler for the device type is assumed.

Standard handlers are as follows (see Appendix C).

TY uses KBTIO	CP uses CRDOUT
PR uses PTAP	LP uses PRTOUT
PP uses PTAP	DC uses DISCIO
CR uses CRDIN	9T, 7T, and MT use MTAP

DEDICATE, value specifies that the device is to be dedicated. Value may be S (system), F (foreground), or B (background). If this option is omitted, the device is assumed to be undedicated.

PAPER, size, width specifies the (hexadecimal) number of printable lines per page (size), and the maximum (decimal) number of characters per line (width). This option applies to typewriters, teletypes, and line printers. If this option is omitted, the values 38_{10} and 132_{10} are assumed for size and width, respectively.

The direct-access disc storage area is dynamically allocated, based on the defined system. The remaining area may be constrained by the following options. (The area not specified otherwise is allocated for foreground storage.) These options are specified in hexadecimal.

1. [(SIZE, value)] [,(NSPT, value)] [,(SS, value)]

   where

   SIZE, value specifies the (hexadecimal) number of disc tracks available to the system. If this option is omitted, the value is assumed to equal the sum of PAS, PFA, PER, FDA, and BCHK (see below).

   NSPT, value specifies the (hexadecimal) number of sectors per track. If this option is omitted, the value 16_{10} is assumed.

   SS, value specifies the (hexadecimal) number of words per sector. If this option is omitted, the value 256_{10} is assumed.

2. (BTRACK, number[, number]...)

   where

   number specifies the (hexadecimal) number of a track that is not to be used by the system.

3. (keyword operand)

   where the keyword operands are:

   PER, value specifies the (hexadecimal) number of tracks to be allocated for peripheral storage (symbiont queue storage). The default value is 200_{10}.

   PFA, value specifies the (hexadecimal) number of tracks to be allocated for permanent file storage (including element files). The default value is 300_{10}.

   PSA, value specifies the (hexadecimal) number of tracks to be allocated for permanent system storage. The default value is 12_{10}.

   BCHK, value specifies the (hexadecimal) number of tracks to be allocated for background checkpoint storage. The default value is 0.

:SDEVICE The :SDEVICE installation control command may be used to specify the device(s) to be associated with a designated symbiont. This control command may be used only immediately following the :DEVICE or :STDLB command. There may not be more than one :SDEVICE command, but continuation cards may be used.

The :SDEVICE installation control command has the form

```
:SDEVICE (LMN, symbiont, name[,...])[,(...)...]
```

where

symbiont specifies the load module name of a symbiont. Either ISSEG (input) or OSSEG (output) may be specified.

name specifies a device name (see Tables 2, 3, and 4 in Chapter 2).

Example:

```
:SDEVICE (LMN,ISSEG,CRA11)
```

This example associates the standard input symbiont with Card Reader 11 on Channel A.

:MONITOR The :MONITOR installation control command may be used to select certain Monitor and CPU options.

The :MONITOR installation control command has the form

```
:MONITOR (option) [,(option)] . . . [,(option)]
```

where the options are

RBLOCK, n specifies that general register block n (decimal) is available to the Monitor.

SFIL, n specifies the maximum (decimal) number of files that can be maintained by symbionts. The default is 20.

TSTACK, size specifies the size, in (decimal) words, of the Monitor's temp stack. The default value is 192.

CORE, size specifies the size, in (decimal) units of K (where K = 1024 words). The default value is 16.

QUEUE, size specifies the maximum (decimal) number of I/O operations that may be queued at any one time. The default value is 4.

MPOOL, size specifies the (decimal) number of 34-word buffers to be pooled for use by the Monitor. The default value is 3. The value specified should be 2 greater than the number of buffered I/O devices used.

SPOOL, size specifies the (decimal) number of 256-word buffers to be pooled for use by the symbionts. The default value is 0.

CPOOL, size specifies the (decimal) number of 40-word buffers to be pooled for symbionts' context blocks. The default value is 0.

CFU, size specifies the (decimal) number of 19-word buffers to be pooled for current file users. The default value is 1.

:RESERVE The :RESERVE installation control command may be used to allocate various areas of core storage.

The :RESERVE installation control command is of the form

```
:RESERVE (option)[,(option)] . . . [,(option)]
```

where the options are

MPATCH, size specifies the (decimal) number of word locations to be reserved for modification and expansion of the Monitor. The default is 0.

RESDF, size specifies the (decimal) number of word locations to be reserved for resident foreground storage. The default is 0.

FIPOOL, value specifies the (decimal) number of 256-word buffers to be pooled for use in foreground file indexing. The default is 0.

FFPOOL, value specifies the (decimal) number of 512-word buffers to be assigned for use in file management (to be used by the Monitor in the packing and unpacking of foreground data). The default is 0.

:DLIMIT The :DLIMIT installation control command may be used to specify the system default limits that are to be associated with each job (see "LIMIT", in Chapter 2).

The :DLIMIT installation control command has the form

```
:DLIMIT [(option)] [,(option)] . . .
```

where the options are

TIME, value specifies the (decimal) default limit for job execution time. Value is expressed in minutes. If unspecified, the value 5 is assumed.

LO, value specifies the (decimal) default limit for the number of pages to be listed by all processors involved in running a job. If unspecified, the value 100 is assumed.

PO, value specifies the (decimal) default limit for the number of object records produced in running a job. If unspecified, the value 500 is assumed.

DO, value specifies the (decimal) default limit for the number of pages of diagnostics produced in running a job. If unspecified, the value 100 is assumed.

UO, value specifies the (decimal) default limit for the number of pages that may be output by the executing program(s) in a job. If unspecified, the value 100 is assumed.

TSTORE, value specifies the (decimal) default limit for the number of granules (512 words) of temporary disc storage that may be used by a job. If unspecified, the value 64 is assumed.

PSTORE, value specifies the (decimal) default limit for the number of granules of permanent disc storage that may be used by a job. If unspecified, the value 64 is assumed.

IPOOL, value specifies the (decimal) default number of 100-word buffers to be pooled for batch file indexing. If unspecified, the value 1 is assumed.

FPOOL, value specifies the (decimal) default number of 512-word file blocking buffers to be allocated to batch tasks. If unspecified, the value 1 is assumed.

:ABS The :ABS installation control command may be used to specify which processors are to be entered into the system in absolute format and the size of an absolute file area for fast access to temporary disc storage. Processors entered in this manner will be managed as part of the system, thereby allowing a direct fetch of the processor. Only one :ABS command may be used, but continuation cards may be used.

The :ABS installation control command has the form

:ABS[, size][(proc$_1$ [, S]) [,(proc$_2$[, S])]. . .]

where

proc$_i$ specifies the name of a processor to be assigned an absolute disc address.

S specifies that the load module form of the processor is to be saved rather than deleted from the system. System Generation will always save the load module form of an overlayed processor that has been declared in the :ABS control command. The root of the tree structure is the only portion of the processor affected by the :ABS control card; therefore, the load module must be saved. System disc storage space requirements are greatly reduced for non-overlayed processors that are not saved.

size is the number of words desired for the absolute storage area on disc.

:INTS The :INTS installation control command may be used to specify what external interrupts are to be connected to resident tasks included in the resident portion of the Monitor.

The :INTS installation control command has the form

:INTS (loc, name, entry[, DIRECT[, rblock]]) [, . . .]

where

loc specifies the location, in hexadecimal, of the designated interrupt.

name specifies the name of a foreground load module.

entry specifies the location to which control is to be transferred when the interrupt (or trap) occurs. The address may be an absolute hexadecimal number or a relative hexadecimal reference.

DIRECT[, rblock] specifies that the interrupt routine is to be entered directly whenever the associated interrupt becomes active. If "DIRECT" is omitted, the interrupt routine will be entered via the MONITOR's interrupt service routine. The option "rblock" specifies that the indicated (hexadecimal) register block may be used by the interrupt routine.

:INTR The :INTR installation control command may be used to specify the load module names of all foreground tasks (other than those specified by :INTS or :TIME commands or included in the resident portion of the Monitor), whether resident or nonresident, and the interrupt location to be associated with each load module. The priority of each task is determined by the priority of its associated interrupt location.

The :INTR installation control command has the form

:INTR (loc, name $\left[,\begin{Bmatrix}\text{RESD}\\ \text{NRESD}\end{Bmatrix}\right]$ [, MASTER]), . . .

where

loc specifies the location, in hexadecimal, of the designated interrupt.

name specifies the name of a foreground load module.

RESD specifies that the designated foreground task is to be loaded into resident core memory.

NRESD specifies that the designated foreground task is to be nonresident. If neither RESD nor NRESD is specified, the default is NRESD.

MASTER specifies that the task may run in the master mode. If MASTER is not specified, the default is the slave mode. If a task is resident and is to be connected directly, MASTER must be specified in the INTR command.

:TIME The :TIME installation control command may be used to define the names of resident foreground tasks to be connected to the standard clock interrupt location, 5B$_{16}$ (i.e., the Monitor's clock).

The :TIME installation control command is of the form

:TIME name$_1$, name$_2$, name$_3$, . . .

where

name specifies the name of a resident foreground task to be connected to the Monitor's clock. The named task is loaded and connected to the Monitor's clock through a normal background job, using the !RUN control command with the "LMN, name" and "TIME, value" parameters (see RUN control command).

!DEF The !DEF (define) control command must be the last command given in System Generation. It causes a complete Monitor system to be formed from the parameters and elements that have been entered.

The DEF processor is responsible for the formation of a bootable system tape with all load modules that were under the account number, under which DEF was run. When bootstrapped from tape, the system will load itself, all load modules, and a disc bootstrap on disc. The completed system is output on the PO device as a Monitor system tape (M:PO having been assigned to an OUTSN tape). This tape begins with a bootstrap loader.

The !DEF installation control command is of the form

```
!DEF [(option)] [,(option)]
```

where the options are

INCL, name [, name] ... specifies that the named files (not load modules) are to be copied on the PO device.

DELETE specifies that all relocatable object modules are to be deleted from the disc storage.

SYSTEM GENERATION EXAMPLE

An example of a typical deck setup for System Generation is shown in Figure 5.

SYSTEM GENERATION MESSAGES

The following messages are output by the various System Generation processors.

PASS-1 MESSAGES

CONTROL COMND ERR

This message is output on the LL device when PASS-1 encounters a control command that is not recognized, contains a syntax error, or has no colon in column 1. PASS-1 makes an error return to the Monitor.

DELIM ERR

This message is output on the LL device when PASS-1 encounters a control command containing an incorrect delimiter. PASS-1 makes an error return to the Monitor.

NAME ERR

This message is output on the LL device when PASS-1 encounters a control command containing an illegal name (i.e., one having no alphabetic character or one having a non-alphanumeric character). PASS-1 makes an error return to the Monitor.

CARD SEQUENCE ERR

This message is output on the LL device when PASS-1 encounters an installation control command that is out of the proper sequence. PASS-1 makes an error return to the Monitor.

NO BO WILL BE GENERATED

This message is output on the LL device if PASS-1 encounters no :SYSWRT command during System Generation. Since this is not an error condition, PASS-1 continues.

FILE NOT ON UPDATE CARD – IGNORED

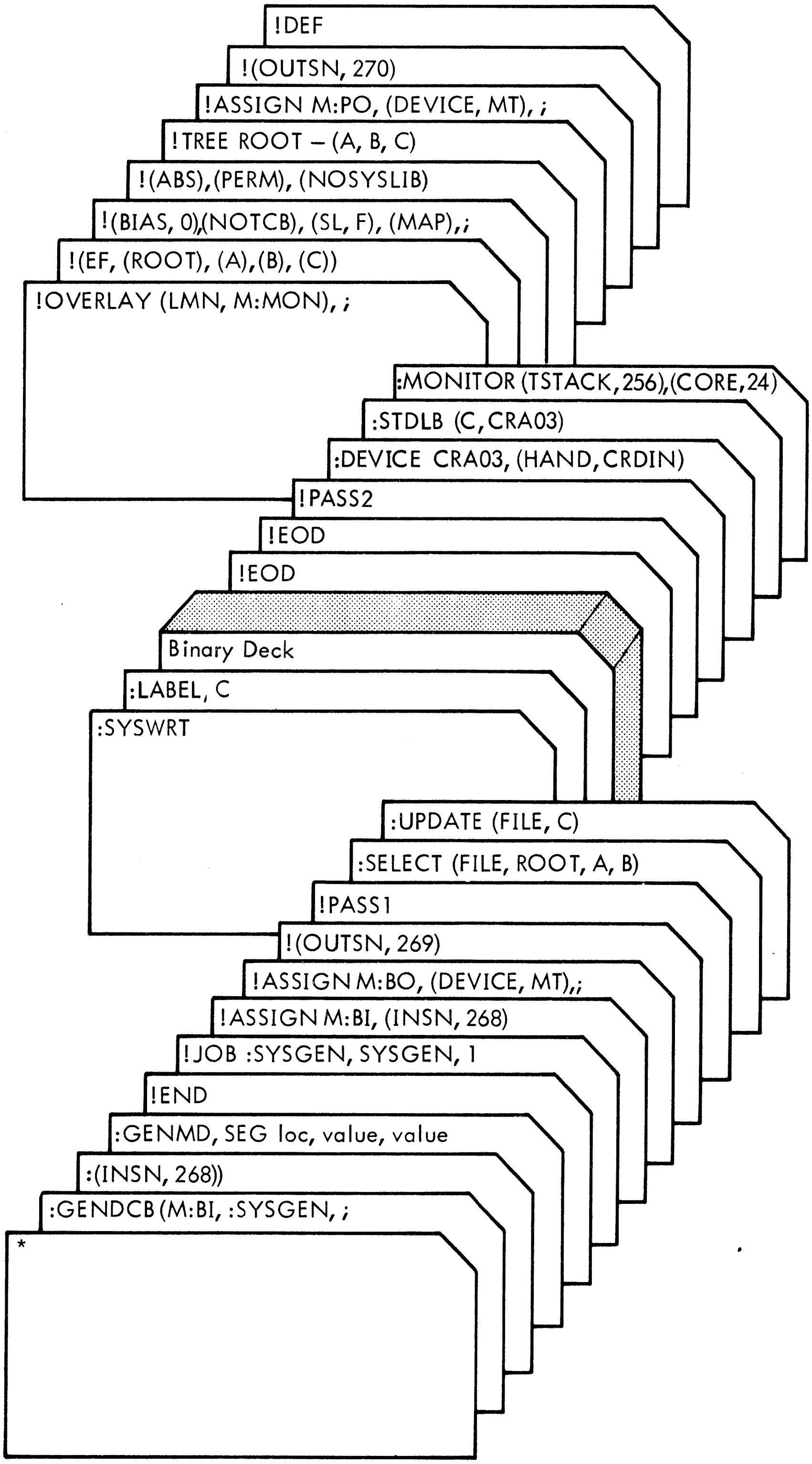
This message is output on the LL device if a file named on a :LABEL card does not appear on a :UPDATE card. PASS-1 ignores the corresponding binary deck from the M:EI device and continues.

IO ABNORMAL

This message is output on the LL device if an I/O error occurs during PASS-1, and an error return to the Monitor is made.

CANNOT FORM BOOTABLE MONITOR

This message is output on the LL device to indicate that a segment of the Monitor is missing or the Monitor master tape cannot be output because of an I/O error. PASS-1 makes an abort return to the Monitor. (This message is also output by the DEF processor if the generated Monitor tape cannot be output.)



```
!DEF
!(OUTSN, 270)
!ASSIGN M:PO, (DEVICE, MT), ;
!TREE ROOT – (A, B, C)
!(ABS),(PERM), (NOSYSLIB)
!(BIAS, 0),(NOTCB), (SL, F), (MAP),;
!(EF, (ROOT), (A),(B), (C))
!OVERLAY (LMN, M:MON), ;
:MONITOR (TSTACK,256),(CORE,24)
:STDLB (C,CRA03)
:DEVICE CRA03, (HAND,CRDIN)
!PASS2
!EOD
!EOD
Binary Deck
:LABEL, C
:SYSWRT
:UPDATE (FILE, C)
:SELECT (FILE, ROOT, A, B)
!PASS1
!(OUTSN, 269)
!ASSIGN M:BO, (DEVICE, MT),;
!ASSIGN M:BI, (INSN, 268)
!JOB :SYSGEN, SYSGEN, 1
!END
:GENMD, SEG loc, value, value
:(INSN, 268))
:GENDCB (M:BI, :SYSGEN, ;
*
```

Figure 5. System Generation Example

PASS-2 (:DEVICE/:STDLB) MESSAGES

```
....PASS-2 CCI IN CONTROL....
```

This message is output on the LL device to indicate that PASS-2 has begun processing.

```
....END OF PASS 2....
```

This message is output on the LL device to indicate that PASS-2 has completed its processing. PASS-2 makes a normal exit to the Monitor.

```
***UNKNOWN CC
```

This message is output on the LL device if PASS-2 encounters a control command that it does not recognize. PASS-2 ignores the unrecognized command and continues processing.

```
**STDLB ENTRY TABLE FULL
```

```
**DEVICE ENTRY TABLE FULL
```

One of these messages is output on the LL device to indicate that there have been too many :STDLB or :DEVICE entries. PASS-2 ignores the supervenient entries and continues.

```
**TYPMNE ENTRY TABLE FULL
```

This message is output on the LL device to indicate that there is not enough core space available for the "mnemonic" entry table. PASS-2 continues.

```
**DISC ENTRY TABLE FULL
```

This message is output on the LL device to indicate that too many discs have been defined. PASS-2 continues.

```
**HANDLER CLIST FULL
```

```
**DCT TABLE FULL
```

```
**HGP TABLE FULL
```

One of these messages is output on the LL device to indicate that not enough core storage is available for the designated table. PASS-2 exits to the Monitor.

```
**DISC OPTIONS MISSING
```

This message is output on the LL device to indicate that the disc definition (:DEVICE command) does not contain sufficient information. PASS-2 exits to the Monitor.

```
**OPLB xx EQUIVALENT yy MISSING
```

This message is output on the LL device to indicate that standard label "yy", to which label "xx" has been equated, has not been defined. PASS-2 continues.

```
**UNKNOWN DEVICE yyndd
```

This message is output on the LL device to indicate that the device code "yyndd" is syntactically incorrect (e.g., "CRXG1") or the device is not available. PASS-2 continues.

```
**INSUFFICIENT PAGES AVAILABLE
```

This message is output on the LL device to indicate that PASS-2 cannot obtain sufficient core storage to work with. PASS-2 exits to the Monitor.

```
**MODIFY ERROR
```

This message is output on the LL device to indicate that some error condition has developed within PASS-2. or the normal core allocation was not sufficient for PASS-2 processing. PASS-2 exits to the Monitor.

```
**NO DISC DEFINED
```

This message is output on the LL device to indicate that no :DEVICE command defining a disc unit was encountered. PASS-2 exits to the Monitor.

```
**NO HANDLER NAME GIVEN
```

This message is output on the LL device to indicate that no handler has been specified for a nonstandard I/O device. PASS-2 continues.

```
**DEVICE TYPE yy ILLEGAL
```

This message is output on the LL device if an illegal device type (not a valid type code or not alphanumeric) has been specified. PASS-2 continues.

PASS-2 (:SDEVICE) MESSAGES

INVALID SYMBIONT NAME

This message is output on the LL device if an illegal character string has been used as a symbiont name. PASS-2 continues.

INVALID KEYWORD

This message is output on the LL device if an invalid keyword (i.e., not "LMN") is encountered in a :SDEVICE command. PASS-2 continues.

SYNTAX ERROR

This message is output on the LL device if a control command contains an error in syntax. PASS-2 continues.

INVALID "yyndd"

This message is output on the LL device if a specified I/O device type is not recognized or is syntactically in error. PASS-2 continues.

NO ROOM LEFT FOR :SDEVICE

This message is output on the LL device to indicate that insufficient core storage is available for processing a :SDEVICE command. PASS-2 makes an abort return to the Monitor.

REMAINDER OF CC IGNORED

This message is output on the LL device to indicate that the unprocessed portion of a control command will be ignored by PASS-2. This message appears in conjunction with other messages, as appropriate. PASS-2 skips to the next command.

MODIFY ERROR

This message is output on the LL device to indicate that some error condition has developed within PASS-2 or the normal core allocation was not sufficient for PASS-2 processing of the :SDEVICE command. PASS-2 makes an abort return to the Monitor.

"SDEVICE" ABORTED

This message is output on the LL device to indicate that PASS-2 has aborted while processing the :SDEVICE command.

PASS-2 (:MONITOR) MESSAGES

ERROR, DEFAULT TAKEN

This message is output on the LL device if a field of a :MONITOR command contains an error in syntax or specifies a value less than the minimum allowed for the designated parameter. PASS-2 continues.

NO PAGES

This message is output on the LL device if insufficient core storage is available to process a :MONITOR command. PASS-2 exits to the Monitor.

MODIFY ERROR – ABORT MON*

This message is output on the LL device if insufficient core storage is available or an error occurs in processing a :MONITOR command. PASS-2 makes an ERR return to the Monitor.

PASS-2 (:ABS) MESSAGES

":L" NAME ILLEGAL OR NAME ALREADY DEFINED

This message is output on the LL device to indicate that a processor name (beginning with ":L") is illegal or has already been defined. PASS-2 continues.

NO PAGES AVAILABLE
"ABS" ABORTED

This message is output on the LL device if an insufficient number of pages of core storage is available for :ABS command processing. PASS-2 makes an abort return to the Monitor.

"(" EXPECTED BUT NOT FOUND

This message is output on the LL device if a field delimiter (a lift parenthesis) was not found in an :ABS command. PASS-2 continues.

```
NO FIELDS ON CC
```

This message is output on the LL device if a :ABS command does not define anything. PASS-2 continues.

```
INVALID PROCESSOR NAME
```

This message is output on the LL device if a :ABS processor name is invalid (contains a non-alphanumeric character or no alphabetic character). PASS-2 continues.

```
"S" EXPECTED BUT NOT FOUND** "S" ASSUMED
```

This message is output on the LL device if PASS-2 expected an "S" to follow a comma in a :ABS command, but no "S" was found. PASS-2 continues.

```
")" EXPECTED BUT NOT FOUND
```

This message is output on the LL device if a field terminator (a right parenthesis) was not found in an :ABS command. PASS-2 continues.

```
SYNTAX ERROR
```

This message is output on the LL device if a syntax error is detected in a :ABS command. PASS-2 continues.

```
PROCESSOR NAME > 11 CHARACTERS
```

This message is output on the LL device if the name of a processor specified in a :ABS command is in excess of 11 characters. PASS-2 continues.

```
INVALID SIZE OR SIZE MISSING, DEFAULT TAKEN
```

This message is output on the LL device to indicate that the contents of the "size" field of a :ABS command are illegal, not defined, or outside the proper range of values. PASS-2 continues.

```
LOAD MODULE GEN. UNSUCCESSFUL
```

This message is output on the LL device if :ABS load module generation is unsuccessful due to errors or insufficient core storage. PASS-2 makes an abort return to the Monitor.

PASS-2 (:DLIMIT) MESSAGES

```
**ERROR, DEFAULT TAKEN
```

This message is output on the LL device if the contents of a numeric :DLIMIT field are illegal or outside the proper range of values. PASS-2 continues.

```
NO PAGES
```

This message is output on the LL device if insufficient core storage is available for processing the :DLIMIT command. PASS-2 makes an abort return to the Monitor.

```
MODIFY ERR – ABORT DLIMIT**
```

This message is output on the LL device if there is an error in processing a :DLIMIT command or if there is insufficient core storage available. PASS-2 makes an abort return to the Monitor.

PASS-2 (:DLIB) MESSAGES

```
***REFERENCES TO UNAVAILABLE FILES–IGNORED**
```

This message is output on the LL device to indicate that a referenced file for a :DLIB definition does not exist. PASS-2 continues.

```
DLIB CONTROL COMMAND IN ERROR– DLIB ABORTED
```

This message is output on the LL device if a :DLIB command contains an error in syntax. PASS-2 makes an abort return to the Monitor.

```
INSUFFICIENT WORKING SPACE–DLIB ABORTED
```

This message is output on the LL device if insufficient core storage is available to process a :DLIB command. PASS-2 makes an abort return to the Monitor.

PASS-2 (FORMLIB) MESSAGES

```
NO PAGES AVAILABLE – ABORT
```

This message is output on the LL device if insufficient core storage is available to complete the execution of the FORMLIB routine. PASS-2 makes an abort return to the Monitor.

PASS-2 (RJITGEN) MESSAGES

```
NO PAGES
```

This message is output on the LL device if insufficient core storage is available to complete the execution of the RJITGEN routine. PASS-2 makes an abort return to the Monitor.

```
MODIFY ERR – ABORT RJITGEN**
```

This message is output on the LL device if an irrecoverable error occurs during execution of the RJITGEN routine or if the requested core storage is insufficient. PASS-2 makes an abort return to the Monitor.

PASS-2 (:RESERVE) MESSAGES

```
**ERROR, DEFAULT TAKEN
```

This message is output on the LL device if a value specified in a :RESERVE command is illegal or outside the acceptable limits. PASS-2 continues.

```
NO PAGES
```

This message is output on the LL device if insufficient core storage is available to process a :RESERVE command. PASS-2 makes an abort return to the Monitor.

```
**MODIFY ERROR – ABORT RESERVE**
```

This message is output on the LL device if an irrecoverable error occurs during the processing of a :RESERVE command or if the requested core storage is insufficient. PASS-2 makes an abort return to the Monitor.

PASS-2 (INTR/INTS) MESSAGES

```
SKIP TO ")"
```

This message is output on the LL device if a syntax error is found in a field of a :INTR or :INTS command. PASS-2 ignores the field and continues.

```
INVALID SYNTAX
```

This message is output on the LL device if a :INTR or :INTS command contains a syntax error. PASS-2 ignores the command and continues.

```
INVALID LOC
```

This message is output on the LL device if the contents of the LOC field of a :INTR or :INTS command are invalid. PASS-2 ignores the field and continues.

```
INVALID NAME
```

This message is output on the LL device if a name in a :INTR or :INTS command is invalid (i.e., too long, having no alphabetic character, or containing a non-alphanumeric character). PASS-2 ignores the name and continues.

```
INVALID KEYWORD
```

This message is output on the LL device if a keyword in a :INTR or :INTS command is not recognized. PASS-2 ignores the keyword and continues.

```
INVALID ENTRY-ADDRESS
```

This message is output on the LL device if a :INTS or :INTR command contains an invalid entry-address field. PASS-2 ignores the field and continues.

```
INVALID RBLOCK
```

This message is output on the LL device if the contents of an RBLOCK field of a :INTR or :INTS command are too large or the field contains a non-numeric character. PASS-2 ignores the field and continues.

```
UNKNOWN DELIMITER
```

This message is output on the LL device if a :INTR or :INTS command contains an invalid terminator. PASS-2 continues.

```
**MODIFY ERROR, ABORT INTSR**
```

This message is output on the LL device if there is an error in processing a :INTS or :INTR command or insufficient core storage is available. PASS-2 makes an abort return to the Monitor.

PASS-2 (:TIME) MESSAGES

```
NO PAGES
```

This message is output on the LL device if insufficient core storage is available for processing the :TIME command. PASS-2 makes an abort return to the Monitor.

**MODIFY ERR – ABORT M:TIME

This message is output on the LL device if an error occurs in processing the :TIME command or the requested core storage is insufficient. PASS-2 makes an abort return to the Monitor.

ILLEGAL NAME

This message is output on the LL device if a name used in a :TIME command is syntactically incorrect (i.e., too long, non-alphanumeric, or without an alphabetic character). PASS-2 makes an abort return to the Monitor.

SYNTAX

This message is output on the LL device if a :TIME command contains a syntax error. PASS-2 makes an abort return to the Monitor.

11. MONITOR BOOTSTRAP PROCEDURE

The following procedure should be used to introduce any version of the Batch Processing Monitor into a machine, regardless of whether the system is then to be used for System Generation or for a normal operating system.

The Monitor system tape is bootstrapped from any available tape drive (7-track or 9-track, depending on system format). While the resident Monitor is being read in from tape, patch cards may be read in from the control (C) device. These are fixed-format cards containing patches to the operating Monitor system. The format of a patch card is as follows.

```
seg,loc,value
```

where

seg is the hexadecimal segment number (see Appendix N) of the Monitor segment that is to be patched.

loc is the relative hexadecimal location of the patch.

value is the absolute hexadecimal value to be inserted.

The patch cards (if any) must be in order by segment number and are terminated by a card having an asterisk (*) in column 1. The asterisk card must be included in the control card deck regardless of whether any patch cards are used.

After the asterisk card is read, the resident Monitor is operational and control is given to the PASS-0 routine. PASS-0 may be thought of as a system initialization routine and is sometimes considered a preliminary part of System Generation, since it defines the environment in which System Generation takes place (if it takes place at all). However, PASS-0 is more closely related to Monitor loading than to System Generation, and is entered regardless of whether System Generation is to be done or not.

The following commands may be read by PASS-0 to define the environment in which the bootstrapped Monitor is to operate. Standard device assignments are in effect at this time and are assumed by PASS-0 unless changed by a :GENOP (see below).

:GENCHN

:GENOP

:GENDCB

:GENSZ

:GENDC

:GENMD

:GENDEF

:GENEXP

:GENDICT

!END

Note that the only required control cards for PASS-0 are the :GENDCB and !END cards. Any or all of the others may be omitted, but the :GENDCB card is required for M:BI to specify the input serial number and account number of the labeled tape (or tapes) containing the system library, processors, and other nonresident portions of the Monitor system. This tape may or may not be the same as the one from which the resident Monitor was bootstrapped.

After the !END card has been read, the labeled tape specified on the :GENDCB card is read in to introduce the processors, etc. to the system. Any modifications specified on :GENMD cards are made and any user-specified initialization routines (see Appendix N) are called.

At this point, PASS-0 outputs the message

```
* * * * * * * * * * * * * * * *
* *       10 070 BPM        * *
* * * * * * * * * * * * * * * *

FIRST AVAILABLE DISC ADDRESS = nnnn
```

(where nnnn indicates the hexadecimal disc address, track/sector) on the OC device and the system enters the wait state.

PASS-0 CONTROL COMMANDS

Control commands recognized by the PASS-0 routine are described below.

:GENCHN The :GENCHN installation control command is used to define the physical peripheral devices that are to be used.

The :GENCHN installation control command has the form

```
:GENCHN yyndd ,...
```

where

yyndd specifies a physical device name (see "STDLB", in Chapter 2).

Example:

```
:GENCHN CRA12,DCEA9,MTG81,MTAFF
```

Defaults are CRA03, LPA02, CPA04, TYA01, 9TA80, and DCAF0.

:GENOP The :GENOP installation control command is used to assign operational labels to peripheral devices. More than one such assignment may be specified in a single :GENOP command.

The :GENOP installation control command is of the form

```
:GENOP (label,type)[,(label,type)] [,...]
```

where

label specifies one of the operational labels listed below

Label	Reference
C	Installation control command input.
LL	Listing log.
LO	Listing output.

type specifies a physical device type (see Table 2).

Example:

```
:GENOP (C,CRA10),(LO,MTA80)
```

:GENDCB The :GENDCB installation control command is used to define the system DCBs associated with tape input or output.

The :GENDCB installation control command has the form

```
:[,(INSN,value[,...)] )]
:GENDCB (dcb name,account ,password ;
```

where

dcb name specifies the name of the DCB that is to be associated with tape I/O. M:BI is the only valid name.

account specifies the account number (up to 8 alphanumeric characters) that is to be associated with the I/O source.

password specifies the password that is to be associated with the I/O source. The password may be from one through eight alphanumeric characters in length.

INSN,value,... specifies the serial numbers (up to 4 characters in length) of the tapes that are to be used as input during PHASE-0. No more than 3 reels may be specified. The first reel specified should contain the first file, etc. INSN must be specified for the M:BI DCB.

Example:

```
:(INSN,001,002))
:GENDCB (M:BI,ACCOUNT1,PASSWORD1.,
```

:GENSZ The :GENSZ installation control command may be used to specify the size of the core memory available to the CPU. If this command is omitted, PASS-0 assumes that the core memory size is 24K words.

The :GENSZ installation control command has the form

```
:GENSZ size
```

where

size specifies the size, in decimal, of core memory, in units of K (where K = 1024 words).

Example:

```
:GENSZ 32
```

:GENDC The :GENDC installation control command is used to specify the disc storage area that is to be made available to PASS-0.

The :GENDC installation control command has the form

```
:GENDC (option)[,(option)]...
```

where the options are

SIZE,value specifies the (decimal) number of disc tracks to be made available. If this option is omitted, 512 is assumed.

NST,value specifies the (decimal) number of sectors per track. If this option is omitted, 16 is assumed.

NSG,value specifies the (decimal) number of sectors per granule. If this option is omitted, 6 is assumed.

FAT,value specifies the first available file track number, in decimal. If this option is omitted, track 0 is assumed.

If the :GENDC installation control command is omitted, all of the default values given above are assumed.

Example:

```
:GENDC (SIZE,93),(NSG,4),(FAT,11),(NST,32)
```

:GENMD The :GENMD installation control command may be used to insert modifications into any segment of any processor input from BI.

The :GENMD installation control command has the form

```
   ┌        ┌┌              ┐┐┐
 : │ ,value ││ +res(name)   │││
   │        ││ +name        │││
   └        └└              ┘┘┘

                            ┌┌              ┐┐
 :GENMD,segment loc,value   ││ +res(name)   ││ ;
                            ││ +name        ││
                            └└              ┘┘
```

where

segment specifies the name of the segment that is to be modified.

loc specifies a relative[t] hexadecimal location or a positive absolute hexadecimal address at which the modification is to be made. If it is an absolute address, it must be preceded by a plus (+).

value specifies the word that is to be inserted, right-justified, at the indicated location. If more than one value is given, they will be inserted into successive locations.

res specifies the address resolution for the external definition (see "name" below).[t] If this option is omitted, word resolution is assumed. When the instruction or data word has been relocated, this parameter (res) determines what the resolution of the word is to be.

res	Specified Resolution
BA	Byte address
HA	Halfword address
WA	Word address
DA	Doubleword address

name specifies the name of an externally defined symbol whose address or value is to be used to relocate the associated value in the data word.

Examples:

```
:GENMD,SEGA+3FA2,+0F0F0F0F
```

```
:GENMD,SEGB LOC1,-3+NAME1
```

```
:GENMD,SEGC LOC2+490,FFFF00+BA(NAME2)
```

```
:-1+DA(NAME4), +FFFFFFFF+WA(NAME5);
```

```
:GENMD,SEGD LOC3-4,10,13+HA(NAME5);
```

:GENDEF The :GENDEF installation control command may be used to equate an external reference (of the System Generation program) to a specified value.

The :GENDEF installation control command has the form

```
                                 ┌┌              ┐┐
 :GENDEF,segment xref,value      ││ +res(name)   ││
                                 ││ +name        ││
                                 └└              ┘┘
```

where

segment specifies the name of the segment in which the external reference occurs.

xref specifies the name of the external reference that is to be equated to a value.

value specifies the value (relative or absolute hexadecimal) to which the external reference is to be equated.

Exampples:

```
:GENDEF,SEGA1  NAMEA1, +3
```

```
:GENDEF,SEGA2  NAMEA2, -FFA
```

```
:GENDEF,SEGA3  NAMEA3, 0+NAMEXX
```

```
:GENDEF,SEGA4  NAMEA4, -2+WA(NAMEZZ)
```

:GENEXP The :GENEXP installation control command may be used to modify a specified location by the insertion of a specified value. It is similar to the :GENMD command, except for a preloaded load module. In such a case, an external reference to the location just modified may be generated.

The :GENEXP installation control command has the form

```
                                  ┌┌              ┐┐
 :GENEXP,segment  loc,value       ││ +res(name)   ││
                                  ││ +name        ││
                                  └└              ┘┘
```

[t] If an overlay segment is to be modified, the external definition must not have been referenced in a "lower" level of the overlay tree.

where

segment specifies the name of the segment that is to be modified.

loc specifies the relative hexadecimal location or absolute hexadecimal address at which the modification is to be made.

value specifies the word that is to be inserted at the indicated location. If the address field of the inserted word is not currently defined, an expression is generated that has the indicated location as its designation (see "res" and "name" below).

res specifies the address resolution for the external definition (see "name" below). If this option is omitted, word resolution is assumed. When the instruction or data word has been relocated, this parameter (res) determines what the resolution of the word is to be.

res	Specified Resolution
BA	Byte address
HA	Halfword address
WA	Word address
DA	Doubleword address

name specifies the name of an externally defined symbol whose address or value is to be used to relocate its associated value in the data word.

Examples:

```
:GENEXP,SEG1  +129A,14
```

```
:GENEXP,SEG2  LODA+3,+0+NAMEA
```

```
:GENEXP,SEG3  LOCB-2,-0+WA(NAMEB)
```

:GENDICT The :GENDICT installation control command may be used to modify a load module's relocation dictionary. It may be used in conjunction with :GENMD and :GENEXP program modifications.

The :GENDICT installation control command has the form

```
:GENDICT,segment  loc,code
```

where

segment specifies the name of the segment containing the location referenced (see "loc", below).

loc specifies the relative[†] hexadecimal location or absolute hexadecimal address for which the dictionary entry is to be modified.

code specifies the applicable relocation parameters (see table below).

Code	Relocation	Resolution	Load Module Bias
0	Address	Byte	Module
1	Address	Halfword	Module
2	Address	Word	Module
3	Address	Doubleword	Module
8	Left half	Doubleword	Module
9	Right half	Doubleword	Module
A	Both halves	Doubleword	Module
E	Absolute	-	-

Examples:

```
:GENDICT,SEG1A  +10FF,0
```

```
:GENDICT,SEG1B  LOC1A,E
```

```
:GENDICT,SEG1C  LOC1B+59F,9
```

PASS-0 MESSAGES

The following messages may be output by the PASS-0 program, on the LL or OC device. PASS-0 either continues its normal operation or initiates an abort return to the Monitor.

```
* * * * * * * * * * * * * * * * * *
* *        10 070 BPM           * *
* * * * * * * * * * * * * * * * * *
FIRST AVAILABLE DISC ADDRESS = nnnn
```

This message is output on the OC device after the Monitor is bootstrapped from a "BI" or "PO" tape. The system then enters the wait state.

The value "nnnn" is the hexadecimal disc address, track/sector.

```
SYSTEN ABORTED (PASS-0)
```

This message is output on the LL device when PASS-0 cannot continue its normal processing.

```
SKIP TO NEXT CC
```

This message is output on the LL device in conjunction with other messages.

[†] If an overlay segment is to be modified, the external definition must not have been referenced in a lower segment of the overlay tree.

```
**SYSGEN PASS-0 IN CONTROL**
```

This message is output on the LL device to indicate that PASS-0 has begun its processing.

```
**SYSGEN PASS-0 COMPLETED**
```

This message is output on the LL device to indicate that PASS-0 in in the final phase of its processing (i.e., the !END card has been read).

```
PASS-0 CONTROL,NO ':'
```

This message is output on the LL device when PASS-0 encounters a command without a colon in column 1. The command is ignored and PASS-0 skips to the next command.

```
UNKNOWN CONTROL COMMAND
```

This message is output on the LL device when PASS-0 encounters a command that it does not recognize. The command is ignored and PASS-0 skips to the next command.

```
*GENDCB* CC SYNTAX ERROR
```

```
*GENMD* CC SYNTAX ERROR
```

```
*GENDEF* CC SYNTAX ERROR
```

```
*GENDICT* CC SYNTAX ERROR
```

```
SYNTAX ERROR
```

One of these messages is output on the LL device when PASS-0 encounters a command containing a syntax error. The part of the command that is affected by the error is ignored and PASS-0 continues processing.

```
SEGMENT-NAME ERROR
```

This message is output on the LL device when PASS-0 encounters a segment name that is in error in some respect. The name is ignored and PASS-0 continues.

```
VALUE-FIELD SYNTAX ERROR
```

```
LOC-FIELD SYNTAX ERROR
```

One of these messages is output on the LL device when PASS-0 encounters a field with a syntax error. The field in question is ignored and PASS-0 continues.

```
NO PAGE AVAILABLE
```

This message is output on the LL device when insufficient core space is available for use by PASS-0. An abort return to the Monitor is made.

```
INVALID YYNDD
```

```
INVALID YY OR DD FIELD
```

```
INVALID KEYWORD OR VALUE
```

One of these messages is output on the LL device when a physical device address is not defined in the system, a device address field is in error syntactically, or the contents of a keyword or value are unknown. The invalid item is ignored and PASS-0 continues.

```
MODIFICATION LOC OR VALUE INVALID
```

This message is output on the LL device when the location and/or value defined in a :GENMD, :GENDEF, etc. command has a valid syntax but is not valid for the segment specified for modification. The location or value may be out of range, for example, or a name reference may not be found in the segment specified. The invalid location or value is ignored and PASS-0 continues. This message is followed by the message shown below.

```
ttttttt LOC = nnnnnnnn s lllll
```

This message is output on the LL device to indicate which command caused the preceding "MODIFICATION LOC OR VALUE INVALID" message. PASS-0 continues normal processing.

The command mnemonic is indicated by "ttttttt"; the name associated with the "LOC" field is "nnnnnnnn" (if relocatable); "lllll" gives the "LOC" field value, i.e., the addend (if relocatable) or value (if absolute); and "s" is the sign of the "LOC" field.

```
PREVIOUS "MODS" IGNORED
```

This message is output on the LL device to indicate that all previously processed :GENMD, :GENEXP, :GENDEF, and :GENDICT commands are abrogated. PASS-0 continues processing.

```
*"OPEN/CLOSE"* INFOR MISSING
```

```
*"READ"* "BI/TM" ABNORMAL
```

```
*"WRITE"* "TM" ABNORMAL
```

One of these messages is output on the LL device to indicate that an irrecoverable I/O failure has occurred in reading from the "BI" tape or in reading from or writing to the system "TM" (i.e., temporary) disc files. An abort return to the Monitor is made.

```
***NO ABS PROCESSORS REQUESTED
```

This is not an error message. It simply calls attention to the fact that no absolute processors have been requested for the system. After outputting this message on the LL device, PASS-0 continues.

```
***PROC**xxx* BIAS BELOW bkgrdll
```

This message is output on the LL device to indicate that the processor named "xxx" is biased below the lower limit of the background area ("bkgrdll"). The offending processor is ignored and PASS-0 continues.

```
***PROC.**xxx* WILL OVERFLOW PSA AREA
***REMAINDER OF ABS PROCESSORS IGNORED
```

This message is output on the LL device to indicate that the processor named "xxx" is too large for the area allocated for permanent RAD storage. The designated processor and all remaining processors are ignored. PASS-0 continues.

```
***ABS DCB/SCRATCH AREA > PSA AREA DEFINED
***SYSGEN PASS-0 ABORTED**
```

This message is output on the LL device to indicate that the area where DCB assignment information is to be written or the area to be used for absolute scratch files exceeds the space allocated for permanent RAD storage. PASS-0 makes an abort return to the Monitor.

```
xxxPROCESSOR ABSOLUTE
```

This message is output on the LL device to indicate that the processor named "xxx" is absolute. PASS-0 continues.

```
PROCESSOR LMN SAVED
```

This message may appear following the message "xxx PROCESSOR ABSOLUTE" to indicate that the load module form of processor "xxx" is saved (automatic for overlayed processors). PASS-0 continues.

```
PROCESSOR LMN RELEASED
```

This message may appear following the message "xxx PROCESSOR ABSOLUTE" to indicate that the load module form of processor "xxx" is released. PASS-0 continues.

```
---- ABS GENERATION COMPLETED
```

This message is output on the LL device to indicate that PASS-0 has completed its function. A normal exit to the Monitor is made.

```
***CANNOT OPEN**xxx*FILE
```

This message is output on the LL device when the processor "xxx" cannot be found in the designated account. PASS-0 then skips to the next processor (if any) to be absolutized.

```
***CANNOT READ KEY**zzz*IN**xxx*
```

This message is output on the LL device when the file named "xxx" is opened but does not contain the record identified by key "zzz". The key for protection type 00 and/or 01 for the designated processor cannot be obtained. PASS-0 continues.

APPENDIX A. 10 070 STANDARD OBJECT LANGUAGE

INTRODUCTION

GENERAL

The CII 10 070 standard object language provides a means of expressing the output of any 10 070 processor in standard format. All programs and subprograms in this object format can be loaded by the Monitor's relocating loader. Such a loader is capable of providing the program linkages needed to form an executable program in core storage. The object language is designed to be both computer-independent and medium-independent; i.e., it is applicable to any CII 10 070 computer having a 32-bit word length, and the same format is used for both cards and paper tape.

SOURCE CODE TRANSLATION

Before a program can be executed by the computer, it must be translated from symbolic form to binary data words and machine instructions. The primary stages of source program translation are accomplished by a processor. However, under certain circumstances, the processor may not be able to translate the entire source program directly into machine language form.

If a source program contains symbolic forward references, a single-pass processor such as the CII Symbol assembler can not resolve such references into machine language. This is because the machine language value for the referenced symbol is not established by a one-pass processor until after the statement containing the forward reference has been processed.

A two-pass processor, such as the CII Meta-Symbol assembler, is capable of making "retroactive" changes in the object program before the object code is output. Therefore, a two-pass processor does not have to output any special object codes for forward references. An example of a forward reference in a Symbol source program is given below.

```
        .
        .
Y       EQU     $ + 3
        .
        .
        CI,5    Z
        .
        .
        LI,R    Z
        .
        .
Z       EQU     2
        .
        .
        BG      Z
        .
        .
R       EQU     Z + 1
        .
        .
```

In this example the operand $ + 3 is not a forward reference because the assembler can evaluate it when processing the source statement in which it appears. However, the operand Z in the statement

```
    CI,5    Z
```

is a forward reference because it appears before Z has been defined. In processing the statement, the assembler outputs the machine-language code for CI,5, assigns a forward reference number (e.g., 12) to the symbol Z, and outputs that forward reference number. The forward reference number and the symbol Z are also retained in the assembler's symbol table.

When the assembler processes the source statement

```
    LI,R    Z
```

it outputs the machine-language code for LI, assigns a forward reference number (e.g., 18) to the symbol R, outputs that number, and again outputs forward reference number 12 for symbol Z.

On processing the source statement

```
    Z    EQU    2
```

the assembler again outputs symbol Z's forward reference number and also outputs the value, which defines symbol Z, so that the relocating loader will be able to satisfy references to Z in statements CI,5 Z and LI,R Z. At this time, symbol Z's forward reference number (i.e., 12) may be deleted from the assembler's symbol table and the defined value of Z equated with the symbol Z (in the symbol table). Then, subsequent references to Z, as in source statement

```
    BG    Z
```

would not constitute forward references, since the assembler could resolve them immediately by consulting its symbol table.

If a program contains symbolic references to externally defined symbols in one or more separately processed subprograms or library routines, the processor will be unable to generate the necessary program linkages.

An example of an external reference in a Symbol source program is shown below.

```
    REF     ALPH
    .
    .
    LI,3    ALPH
    .
    .
```

When the assembler processes the source statement

```
    REF    ALPH
```

it outputs the symbol ALPH, in symbolic (EBCDIC) form, in a declaration specifying that the symbol is an external reference. At this time, the assembler also assigns a declaration name number to the symbol ALPH but does not output the number. The symbol and name number are retained in the assembler's symbol table.

After a symbol has been declared an external reference, it may appear any number of times in the symbolic subprogram in which it was declared. Thus, the use of the symbol ALPH in the source statement

```
LI,3     ALPH
```

in the above example, is valid even though ALPH is not defined in the subprogram in which it is referenced.

The relocating loader is able to generate interprogram linkages for any symbol that is declared an external definition in the subprogram in which that symbol is defined. Shown below is an example of an external definition in a Symbol source program.

```
           DEF      ALPH
            .
            .
           LI,3     ALPH
            .
            .
ALPH       AI,4     X'F2'
            .
            .
```

When the assembler processes the source statement

```
DEF     ALPH
```

it outputs the symbol ALPH, in symbolic (EBCDIC) form, in a declaration specifying that the symbol is an external definition. At this time, the assembler also assigns a declaration name number to the symbol ALPH but does not output the number. The symbol and name number are retained in the assembler's symbol table.

After a symbol has been declared an external definition it may be used (in the subprogram in which it was declared) in the same way as any other symbol. Thus, if ALPH is used as a forward reference, as in the source statement

```
LI,3        ALPH
```

above, the assembler assigns a forward reference number to ALPH, in addition to the declaration name number assigned previously. (A symbol may be both a forward reference and an external definition.)

On processing the source statement

```
ALPH        A1,4      X'F2'
```

the assembler outputs the declaration name number of the label ALPH (and an expression for its value) and also outputs the machine-language code for AI,4 and the constant X'F2'.

OBJECT LANGUAGE FORMAT

An object language program generated by a processor is output as a string of bytes representing "load items". A load item consists of an item type code followed by the specific load information pertaining to that item. (The detailed format of each type of load item is given later in this appendix.) The individual load items require varying numbers of bytes for their representation, depending on the type and specific content of each item. A group of 108 bytes, or fewer, comprises a logical record. A load item may be continued from one logical record to the next.

The ordered set of logical records that a processor generates for a program or subprogram is termed an "object module". The end of an object module is indicated by a module-end type code followed by the error severity level assigned to the module by the processor.

RECORD CONTROL INFORMATION

Each record of an object module consists of 4 bytes of control information followed by a maximum of 104 bytes of load information. That is, each record, with the possible exception of the end record, normally consists of 108 bytes of information (i.e., 72 card columns).

The 4 bytes of control information for each record have the form and sequence shown below.

Byte 0

Record Type			Mode		Format		
			1	1	1	0	0
0	1	2	3	4	5	6	7

Byte 1

Sequence Number	
0	7

Byte 2

Checksum	
0	7

Byte 3

Record Size	
0	7

Record Type   specifies whether this record is the last record of the module:

000 means last
001 means not last

Mode   specifies that the loader is to read binary information. This code is always 11.

Format   specifies object language format. This code is always 100.

Sequence Number   is 0 for the first record of the module and is incremented by 1 for each record thereafter, until it recycles to 0 after reaching 255.

Checksum   is the computed sum of the bytes comprising the record. Carries out of the most significant bit position of the sum are ignored.

Record Size   is the number of bytes (including the record control bytes) comprising the logical record (5 ≤ record size ≤ 108). The recordsize will normally be 108 bytes for all records except the last one, which may be fewer. Any excess bytes in a physical record are ignored.

LOAD ITEMS

Each load item begins with a control byte that indicates the item type. In some instances, certain parameters are also provided in the load item control byte. In the following discussion, load items are categorized according to their function:

1. Declarations identify to the loader the external and control section labels that are to be defined in the object module being loaded.
2. Definitions define the value of forward references, external definitions, the origin of the subprogram being loaded, and the starting address (e.g., as provided in a Symbol/Meta-Symbol END directive).
3. Expression evaluation load items within a definition provide the values (such as constants, forward references, etc.) that are to be combined to form the final value of the definition.
4. Loading items cause specified information to be stored into core memory.
5. Miscellaneous items comprise padding bytes and the module-end indicator.

DECLARATIONS

In order for the loader to provide the linkage between subprograms, the processor must generate for each external reference or definition a load item, referred to as a "declaration", containing the EBCDIC code representation of the symbol and the information that the symbol is either an external reference or a definition (thus, the loader will have access to the actual symbolic name).

Forward references are always internal references within an object module. (External references are never considered forward references.) The processor does not generate a declaration for a forward reference as it does for externals; however, it does assign name numbers to the symbols referenced.

Declaration name numbers (for control sections and external labels) and forward reference name numbers apply only within the object module in which they are assigned. They have no significance in establishing interprogram linkages, since external references and definitions are correlated by matching symbolic names. Hence, name numbers used in any expressions in a given object module always refer to symbols that have been declared within that module.

The processor must generate a declaration for each symbol that identifies a program section. Although the CII Symbol assembler used with the Monitor allows only a standard control section (i.e., program section), the standard object language includes provision for other types of control sections (such as dummy control sections). Each object module produced by the Symbol processor is considered to consist of at least one control section. If no section is explicitly identified in a Symbol source program, the assembler assumes it to be a standard control section (discussed below). The standard control section is always assigned a declaration name number of 0. All other control sections (i.e., produced by a processor capable of declaring other control sections) are assigned declaration name numbers (1, 2, 3, etc.) in the order of their appearance in the source program.

In the load items discussed below, the access code, pp, designates the memory protection class that is to be associated with the control section. The meaning of this code is given below.

pp	Memory Protection Feature[†]
00	Read, write, or access instructions from.
01	Read or access instructions from.
10	Read only.
11	No access.

Control sections are always allocated on a doubleword boundary. The size specification designates the number of bytes to be allocated for the section.

Declare Standard Control Section

Byte 0

Control byte							
0	0	0	0	1	0	1	1
0	1	2	3	4	5	6	7

Byte 1

Access code				Size (bits 1 through 4)			
p	p	0	0				
0	1	2	3	4	5	6	7

Byte 2

Size (bits 5 through 12)	
0	7

Byte 3

Size (bits 13 through 20)	
0	7

This item declares the standard control section for the object module. There may be no more than one standard control section in each object module. The origin of the standard control section is effectively defined when the first reference to the standard control section occurs, although the declaration item might not occur until much later in the object module.

[†] "Read" means a program can obtain information from the protected area; "write" means a program can store information into a protected area; and, "access" means the computer can execute instructions stored in the protected area.

This capability is obviously required by one-pass processors, since the size of a section cannot be determined until all of the load information for that section has been generated by the processor.

Declare Non-Standard Control Section

Byte 0

Control byte							
0	0	0	0	1	1	0	0
0	1	2	3	4	5	6	7

Byte 1

Access code				Size (bits 1 through 4)			
P	P	0	0				
0	1	2	3	4			7

Byte 2

Size (bits 5 through 12)	
0	7

Byte 3

Size (bits 13 through 20)	
0	7

This item declares a control section other than standard control section (see above). Note that this item is not applicable to the CII Symbol processor used with the Monitor system. However, the loader is capable of loading object modules (produced by other processors, such as the Meta-Symbol and FORTRAN IV processors) that do contain this item.

Declare Dummy Section

Byte 0

Control byte							
0	0	0	0	1	0	0	1
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

Byte 3

Access code				Size (bits 1 through 4)			
P	P	0	0				
0	1	2	3	4			7

Byte 4

Size (bits 5 through 12)	
0	7

Byte 5

Size (bits 13 through 20)	
0	7

This item comprises a declaration for a dummy control section. It results in the allocation of the specified dummy section, if that section has not been allocated previously by another object module. The label that is to be associated with the first location of the allocated section must be a previously declared external definition name. (Even though the source program may not be required to explicitly designate the label as an external definition, the processor must generate an external definition name declaration for that label prior to generating this load item.)

Declare External Definition Name

Byte 0

Control byte							
0	0	0	0	0	0	1	1
0	1	2	3	4	5	6	7

Byte 1

Name length, in bytes (K)	
0	7

Byte 2

First byte of name	
0	7

⋮

Byte K+1

Last byte of name	
0	7

This item declares a label (in EBCDIC code) that is an external definition within the current object module. The name may not exceed 63 bytes in length.

Declare Primary External Reference Name

Byte 0

Control byte							
0	0	0	0	0	1	0	1
0	1	2	3	4	5	6	7

Byte 1

Name length (K), in bytes	
0	7

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

Byte 2

First byte of name	
0	7

⋮

Byte K+1

Last byte of name	
0	7

This item declares a symbol (in EBCDIC code) that is a primary external reference within the current object module. The name may not exceed 63 bytes in length.

A primary external reference is capable of causing the loader to search the system library for a corresponding external definition. If a corresponding external definition is not found in another load module of the program or in the system library, a load error message is output and the job is errored.

Declare Secondary External Reference Name

Byte 0

Control byte							
0	0	0	0	0	1	1	0
0	1	2	3	4	5	6	7

Byte 1

Name length, in bytes (K)	
0	7

Byte 2

First byte of name	
0	7

⋮

Byte K+1

Last byte of name	
0	7

This item declares a symbol (in EBCDIC code) that is a secondary external reference within the current object module. The name may not exceed 63 bytes in length.

A secondary external reference is not capable of causing the loader to search the system library for a corresponding external definition. If a corresponding external definition is not found in another load module of the program, the job is not errored and no error or abnormal message is output.

Secondary external references often appear in library routines that contain optional or alternative subroutines, some of which may not be required by the user's program. By the use of primary external references in the user's program, the user can specify that only those subroutines that are actually required by the current job are to be loaded. Although secondary external references do not cause loading from the library, they do cause linkages to be made between routines that are loaded.

DEFINITIONS

When a source language symbol is to be defined (i.e., equated with a value), the processor provides for such a value by generating an object language expression to be evaluated by the loader. Expressions are of variable length, and terminate with an expression-end control byte (see Section 4 of this appendix). An expression is evaluated by the addition or subtraction of values specified by the expression.

Since the loader must derive values for the origin and starting address of a program, these also require definition.

Origin

Byte 0

Control byte							
0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7

This item sets the loader's load-location counter to the value designated by the expression immediately following the origin control byte. This expression must not contain any elements that cannot be evaluated by the loader (see Expression Evaluation which follows).

Forward Reference Definition

Byte 0

Control byte							
0	0	0	0	1	0	0	0
0	1	2	3	4	5	6	7

Byte 1

First byte of reference number	
0	7

Byte 2

Second byte of reference number	
0	7

This item defines the value (expression) for a forward reference. The referenced expression is the one immediately following byte 2 of this load item, and must not contain any elements that cannot be evaluated by the loader (see Expression Evaluation which follows).

Forward Reference Definition and Hold

Byte 0

Control byte							
0	0	0	1	0	0	0	0
0	1	2	3	4	5	6	7

Byte 1

First byte of reference number	
0	7

Byte 2

Second byte of reference number	
0	7

This item defines the value (expression) for a forward reference and notifies the loader that this value is to be retained in the loader's symbol table until the module end is encountered. The referenced expression is the one immediately following the name number. It may contain values that have not been defined previously, but all such values must be available to the loader prior to the module end.

After generating this load item, the processor need not retain the value for the forward reference, since that responsibility is then assumed by the loader. However, the processor must retain the symbolic name and forward reference number assigned to the forward reference (until module end).

External Definition

Byte 0

Control byte							
0	0	0	0	1	0	1	0
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

This item defines the value (expression) for an external definition name. The name number refers to a previously declared definition name. The referenced expression is the one immediately following the name number.

Define Start

Byte 0

Control byte							
0	0	0	0	1	1	0	1
0	1	2	3	4	5	6	7

This item defines the starting address (expression) to be used at the completion of loading. The referenced expression is the one immediately following the control byte.

EXPRESSION EVALUATION

A processor must generate an object language expression whenever it needs to communicate to the loader one of the following:

1. A program load origin.
2. A program starting address.
3. An external definition value.
4. A forward reference value.
5. A field definition value.

Such expressions may include sums and differences of constants, addresses, and external or forward reference values that, when defined, will themselves be constants or addresses.

After initiation of the expression mode, by the use of a control byte designating one of the five items described above, the value of an expression is expressed as follows:

1. An address value is represented by an offset from the control section base plus the value of the control section base.
2. The value of a constant is added to the accumulated sum by generating an Add Constant (see below) control byte followed by the value, right-justified in four bytes.

   The offset from the control section base is given as a constant representing the number of units of displacement from the control section base, at the resolution of the address of the item. That is, a word address would have its constant portion expressed as a count of the number of words offset from the base, while the constant portion of a byte address would be expressed as the number of bytes offset from the base.

   The control section base value is accumulated by means of an Add Value of Declaration (see below) or Subtract Value of Declaration load item specifying the desired resolution and the declaration number of the control section base. The loader adjusts the base value to the specified address resolution before adding it to the current partial sum for the expression.

   In the case of an absolute address, an Add Absolute Section (see below) or Subtract Absolute Section control byte must be included in the expression to identify the value as an address and to specify its resolution.
3. An external definition or forward reference value is included in an expression by means of a load item adding or subtracting the appropriate declaration or forward reference value. If the value is an address, the resolution specified in the control byte is used to align the value before adding it to the current partial sum for the expression. If the value is a constant, no alignment is necessary.

Expressions are not evaluated by the loader until all required values are available. In evaluating an expression, the loader maintains a count of the number of values added or subtracted at each of the four possible resolutions. A separate counter is used for each resolution, and each counter is incremented or decremented by 1 whenever a value of the corresponding resolution is added to or subtracted from the loader's expression accumulator. The final accumulated sum is a constant, rather than an address value, if the final count in all four counters is equal to 0. If the final count in one (and only one) of the four counters is equal to +1 or -1, the

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

accumulated sum is a "simple address" having the resolution of the nonzero counter. If more than one of the four counters have a nonzero final count, the accumulated sum is termed a "mixed -resolution expression" and is treated as a constant rather than an address.

The resolution of a simple address may be altered by means of a Change Expression Resolution (see below) control byte. However, if the current partial sum is either a constant or a mixed-resolution value when the Change Expression Resolution control byte occurs, then the expression resolution is unaffected.

Note that the expression for a program load origin or starting address must resolve to a simple address, and the single nonzero resolution counter must have a final count of +1 when such expressions are evaluated.

In converting a byte address to a word address, the two least significant bits of the address are truncated. Thus, if the resulting word address is later changed back to byte resolution, the referenced byte location will then be the first byte (byte 0) of the word.

After an expression has been evaluated, its final value is associated with the appropriate load item.

In the following diagrams of load item formats, RR refers to the address resolution code. The meaning of this code is given in the table below.

RR	Address Resolution
00	Byte
01	Halfword
10	Word
11	Doubleword

The load items discussed in this appendix, "Expression Evaluation", may appear only in expressions.

Add Constant

Byte 0

Control byte							
0	0	0	0	0	0	0	1
0	1	2	3	4	5	6	7

Byte 1

First byte of constant

0 7

Byte 2

Second byte of constant

0 7

Byte 3

Third byte of constant

0 7

Byte 4

Fourth byte of constant

0 7

This item causes the specified 4-byte constant to be added to the loader's expression accumulator. Negative constants are represented in two's complement form.

Add Absolute Section

Byte 0

Control byte							
0	0	1	1	0	1	R	R
0	1	2	3	4	5	6	7

This item identifies the associated value (expression) as a positive absolute address. The address resolution code, RR, designates the desired resolution.

Subtract Absolute Section

Byte 0

Control byte							
0	0	1	1	1	0	R	R
0	1	2	3	4	5	6	7

This item identifies the associated value (expression) as a negative absolute address. The address resolution code, RR, designates the desired resolution.

Add Value of Declaration

Byte 0

Control byte							
0	0	1	0	0	0	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of name number

0 7

Byte 2

Second byte of name number[†]

0 7

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

This item causes the value of the specified declaration to be added to the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the name number refers to a previously declared definition name that is to be associated with the first location of the allocated section.

One such item must appear in each expression for a relocatable address occurring within a control section, adding the value of the specified control section declaration (i.e., adding the byte address of the first location of the control section).

Add Value of Forward Reference

Byte 0

Control byte							
0	0	1	0	0	1	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of forward reference number	
0	7

Byte 2

Second byte of forward reference number	
0	7

This item causes the value of the specified forward reference to be added to the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the designated forward reference must not have been defined previously.

Subtract Value of Declaration

Byte 0

Control byte							
0	0	1	0	1	0	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]

This item causes the value of the specified declaration to be subtracted from the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the name number refers to a previously declared definition name that is to be associated with the first location of the allocated section.

Subtract Value of Forward Reference

Byte 0

Control byte							
0	0	1	0	1	1	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of forward reference number	
0	7

Byte 2

Second byte of forward reference number	
0	7

This item causes the value of the specified forward reference to be subtracted from the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the designated forward reference must not have been defined previously.

Change Expression Resolution

Byte 0

Control byte							
0	0	1	1	0	0	R	R
0	1	2	3	4	5	6	7

This item causes the address resolution in the expression to be changed to that designated by RR.

Expression End

Byte 0

Control byte							
0	0	0	0	0	0	1	0
0	1	2	3	4	5	6	7

This item identifies the end of an expression (the value of which is contained in the loader's expression accumulator).

LOADING

Load Absolute

Byte 0

Control byte							
0	1	0	0	N	N	N	N
0	1	2	3	4	5	6	7

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

Byte 1

First byte to be loaded	
0	7

⋮

Byte NNNN

Last byte to be loaded	
0	7

This item causes the next NNNN bytes to be loaded absolutely (NNNN is expressed in natural binary form, except that 0000 is interpreted as 16 rather than 0). The load location counter is advanced appropriately.

Load Relocatable (Long Form)

Byte 0

Control byte							
0	1	0	1	Q	C	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

This item causes a 4-byte word (immediately following this load item) to be loaded, and relocates the address field according to the address resolution code, RR. Control bit C designates whether relocation is to be relative to a forward reference (C = 1) or relative to a declaration (C = 0). Control bit Q designates whether a 1-byte (Q = 1) or a 2-byte (Q = 0) name number follows the control byte of this load item.

If relocation is to be relative to a forward reference, the forward reference must not have been defined previously. When this load item is encountered by the loader, the load location counter can be aligned with a word boundary by loading the appropriate number of bytes containing all zeros (e.g., by means of a load absolute item).

Load Relocatable (Short Form)

Byte 0

Control byte							
1	C	D	D	D	D	D	D
0	1	2	3	4	5	6	7

This item causes a 4-byte word (immediately following this load item) to be loaded, and relocates the address field (word resolution). Control bit C designates whether relocation is to be relative to a forward reference (C = 1) or relative to a declaration (C = 0). The binary number DDDDDD is the forward reference number or declaration number by which relocation is to be accomplished.

If relocation is to be relative to a forward reference, the forward reference must not have been defined previously. When this load item is encountered by the loader, the load location counter must be on a word boundary (see "Load Relocatable (Long Form)", above).

Repeat Load

Byte 0

Control byte							
0	0	0	0	1	1	1	1
0	1	2	3	4	5	6	7

Byte 1

First byte of repeat count	
0	7

Byte 2

Second byte of repeat count	
0	7

This item causes the loader to repeat (i.e., perform) the subsequent load item a specified number of times. The repeat count must be greater than 0, and the load item to be repeated must follow the repeat load item immediately.

Define Field

Byte 0

Control byte							
0	0	0	0	0	1	1	1
0	1	2	3	4	5	6	7

Byte 1

Field location constant, in bits (K)	
0	7

Byte 2

Field length, in bits (L)	
0	7

This item defines a value (expression) to be added to a field in previously loaded information. The field is of length L (1 ≤ L ≤ 255) and terminates in bit position T, where:

T = current load bit position −256 +K.

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

The field location constant, K, may have any value from 1 to 255. The expression to be added to the specified field is the one immediately following byte 2 of this load item.

MISCELLANEOUS LOAD ITEMS

Padding

Byte 0

Control byte							
0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

Padding bytes are ignored by the loader. The object language allows padding as a convenience for processors.

Module End

Byte 0

Control byte							
0	0	0	0	1	1	1	0
0	1	2	3	4	5	6	7

Byte 1

Severity level							
0	0	0	0	E	E	E	E
0	1	2	3	4	5	6	7

This item identifies the end of the object module. The value EEEE is the error severity level assigned to the module by the processor (see "LOAD", in Chapter 2 of this manual).

OBJECT MODULE EXAMPLE

The following example shows the correspondence between the statements of a Symbol source program and the string of object bytes output for that program by the assembler. The program, listed below, has no significance other than illustrating typical object code sequences.

Example

```
 1                                      DEF      AA,BB,CC   CC IS UNDEFINED BUT CAUSES NO
                                                            ERROR
 2                                      REF      RZ,RTN     EXTERNAL REFERENCES DECLARED
 3  00000                      ALPHA    CSECT               DEFINE CONTROL SECTION ALPHA
 4  000C8                               ORG      200        DEFINE ORGIN
 5  000C8      22000000   N    AA       LI,CNT   0          DEFINES EXTERNAL AA; CNT IS A
                                                            FWD REF
 6  000C9      32000000   N             LW,R     RZ         R IS A FORWARD REFERENCE;
 7                             *                            RZ IS AN EXTERNAL REFERENCE, AS
 8                             *                            DECLARED IN LINE 2
 9  000CA      50000000   N    RPT      AH,R     KON        DEFINES RPT; R AND KON ARE
10                             *                            FORWARD REFERENCES
11  000CB      69200000   F             BCS,2    BB         BB IS AN EXTERNAL DEFINITION
12                             *                            USED AS A FORWARD REFERENCE
13  000CC      20000001   N             AI,CNT   1          CNT IS A FORWARD REFERENCE
14  000CD      680000CA                 B        RPT        RPT IS A BACKWARD REFERENCE
15  000CE      68000000   X             B        RTN        RTN IS AN EXTERNAL REFERENCE
16  000CF      0001       A    KON      DATA,2   1          DEFINES KON
17          00000003           R        EQU      3          DEFINES R
18          00000004           CNT      EQU      4          DEFINES CNT
19  000D0      224FFFFF   A    BB       LI,CNT   -1         DEFINES EXTERNAL BB THAT HAS
20                             *                            ALSO BEEN USED AS A FORWARD
21                             *                            REFERENCE
22  000C8                               END      AA         END OF PROGRAM
```

The object string generated for this program is given below.

Record-control information	00111100	Begin record
	00000000	0 (record sequence number)
	01100011	63_{16} (checksum)
	01101100	$6C_{16}$ (number of bytes in record)
Declare AA (source line 1)	00000011	Begin external-definition declaration
	00000010	2 (a 2-byte name follows)
	11000001	$C1_{16}$ (EBCDIC "A")
	11000001	$C1_{16}$ (EBCDIC "A")
Declare BB (source line 1)	00000011	Begin external-definition declaration
	00000010	2 (a 2-byte name follows)
	11000010	$C2_{16}$ (EBCDIC "B")
	11000010	$C2_{16}$ (EBCDIC "B")
Declare CC (source line 1)	00000011	Begin external-definition declaration
	00000010	2 (a 2-byte name follows)
	11000011	$C3_{16}$ (EBCDIC "C")
	11000011	$C3_{16}$ (EBCDIC "C")
Declare RZ (source line 2)	00000101	Begin primary-external-reference declaration
	00000010	2 (a 2-byte name follows)
	11011001	$D9_{16}$ (EBCDIC "R")
	11101001	$E9_{16}$ (EBCDIC "Z")
Declare RTN (source line 2)	00000101	Begin primary-external-reference declaration
	00000011	3 (a 3-byte name follows)
	11011001	$D9_{16}$ (EBCDIC "R")
	11100011	$E3_{16}$ (EBCDIC "T")
	11010101	$D5_{16}$ (EBCDIC "N")
Define AA (source line 5)[†]	00001010	Begin definition of external definition
	00000001	1 (the following expression defines declaration 1, i.e., AA)
	00000001	Add the following 4-byte constant
	00000000	320_{16} (the relocatable byte address of AA)
	00000000	
	00000011	
	00100000	
	00100000	Add the value of the following declaration
	00000000	0 (declaration 0, i.e., the standard control section)
	00000010	End expression mode

[†] No object code is generated for source line 3 (define control section) or 4 (define origin) at the time they are encountered. The control section is declared at the end of the program after Symbol has determined the number of bytes the program requires. The origin definition is generated prior to the first instruction.

Define origin (source line 4)	00000100	Begin definition of origin
	00000001	Add the following 5-byte constant
	00000000	320_{16} (the relocatable byte address of the origin)
	00000000	
	00000011	
	00100000	
	00100000	Add the value of the following declaration
	00000000	0 (declaration 0, i.e., the standard control section)
	00000010	End expression mode
Load code for LI (source line 5)	01000100	Load the next 4 bytes absolutely
	00100010	22000000_{16} (operation code for LI 0)
	00000000	
	00000000	
	00000000	
Define field CNT (source line 5)	00000111	Begin field definition
	11101011	EB_{16} (field-location constant)
	00000100	4 (number of bits in field)
	00100110	Add the value of the following forward reference
	00000000	0 (forward reference 0, i.e., CNT)[†]
	00000000	
	00000010	End expression mode
Load code for LW (source line 6)	10000100	Load the next 4 bytes (word resolution) relative to declaration 4 (i.e., RZ)
	00110010	32000000_{16} (operation code for LW)
	00000000	
	00000000	
	00000000	
Define field R (source line 6)	00000111	Begin field definition
	11101011	EB_{16} (field-location constant)
	00000100	4 (number of bits in field)
	00100110	Add the value of the following forward reference
	00000000	6 (forward reference 6, i.e., R)[†]
	00000110	
	00000010	End expression mode
Load code for AH (source line 9)	11001100	Load the next 4 bytes relative to forward reference C_{16} (i.e., KON)[†]
	01010000	50000000_{16} (operation code for AH)
	00000000	
	00000000	
	00000000	

[†] Forward reference numbers are normally assigned by the Symbol assembler in the sequence 0, 6, C, 12, etc. (hexadecimal).

Define field R (source line 9)	00000111	Begin field definition
	11101011	EB$_{16}$ (field-location constant)
	00000100	4 (number of bits in field)
	00100110	Add the value of the following forward references
	00000000	6 (forward reference 6, i.e., R)
	00000110	
	00000010	End expression mode
Load code for BCS, 2 (source line 11)	11010010	Load the next 4 bytes relative to forward reference 12$_{16}$ (i.e., BB)
	01101001	69200000$_{16}$ (operation code for BCS, 2)
	00100000	
	00000000	
	00000000	
Load code for AI 1 (source line 13)	01000100	Load the next 4 bytes absolutely
	00100000	20000001$_{16}$ (operation code for AI 1)
	00000000	
	00000000	
	00000001	
Define field CNT (source line 13)	00000111	Begin field definition
	00010011	EB$_{16}$ (field-location constant)
	00000100	4 (number of bits in field)
	00100110	Add the value of the following forward reference
	00000000	0 (forward reference 0, i.e., CNT)
	00000000	
	00000010	End expression mode
Load code for B RPT (source line 14)	10000000	Load the next 4 bytes relative to declaration 0 (i.e., the standard control section)
	01101000	68000CA$_{16}$ (code for BCR RPT, i.e., CA$_{16}$ is the relocatable word address of RPT)
	00000000	
	00000000	
	11001010	
Load code for B (source line 15)	10000101	Load the next 4 bytes relative to declaration 5 (i.e., RTN)
	01101000	68000000$_{16}$ (operation code for BCR)
	00000000	
	00000000	
	00000000	
(source line 16)	00001000	Begin forward reference definition (n.b., this item is continued in the next record)

Record-control information	00011100	Begin last record of module
	00000001	1 (record sequence number)
	11101100	EC$_{16}$ (checksum)
	01010001	51$_{16}$ (number of bytes in record)
Load code for DATA, 2 1 (source line 16)	00000000	C$_{16}$ (forward reference C$_{16}$, i.e., KON)
	00001100	
	00000001	Add the following 4-byte constant
	00000000	33C$_{16}$ (the relocatable byte address of KON)
	00000000	
	00000011	
	00111100	
	00100000	Add the value of the following declaration
	00000000	0 (declaration 0, i.e., the standard control section)
	00000010	End expression mode
Define R (source line 17)	00001000	Begin forward reference definition
	00000000	6 forward reference 6, i.e., R)
	00000110	
	00000001	Add the following 4-byte constant
	00000000	3_{16}
	00000000	
	00000000	
	00000011	
	00000010	End expression mode
Define CNT (source line 18)	00001000	Begin forward reference definition
	00000000	0 (forward reference 0, i.e., CNT)
	00000000	
	00000001	Add the following 4-byte constant
	00000000	4_{16}
	00000000	
	00000000	
	00000100	
	00000010	End expression mode
Load 0000$_{16}$ (advance to next word boundary)	00001111	Repeat the following load item as specified
	00000000	2 (the following load item is to be loaded twice)
	00000010	
	01000001	Load the next byte absolutely
	00000000	00$_{16}$ (not a padding control byte)

Define forward reference BB (source line 19)	00001000	Begin forward reference definition
	00000000	12_{16} (forward reference 12_{16}, i.e., BB)
	00010010	
	00000001	Add the following 4-byte constant
	00000000	340_{16} (the relocatable byte address of BB)
	00000000	
	00000011	
	01000000	
	00100000	Add the value of the following declaration
	00000000	0 (declaration 0, i.e., standard control section)
	00000010	End expression mode
Define external definition BB (source line 19)	00001010	Begin definition of external definition
	00000010	2 (the following expression defines declaration 2, i.e., BB)
	00000001	Add the following 4-byte constant
	00000000	340_{16} (the relocatable byte address of BB)
	00000000	
	00000011	
	01000000	
	00100010	Add the value of the following declaration
	00000000	0 (declaration 0, i.e., the standard control section)
	00000010	End expression mode
Load code for LI, CNT -1 (source line 19)	01000100	Load the next 4 bytes absolutely
	00100010	$224FFFFF_{16}$ (code for LI, CNT -1)
	01001111	
	11111111	
	11111111	
Define start (source line 22)	00001101	Begin definition of start
	00000001	Add the following 4-byte constant
	00000000	320_{16} (the relocatable byte address of the start, i.e., AA)
	00000000	
	00000011	
	00100000	
	00100000	Add the value of the following declaration
	00000000	0 (declaration 0; i.e., the standard control section)
	00000010	End expression mode
Declare standard control section	00001011	Begin standard control section declaration
	00000000	344_{16} (access code 00 and 344_{16} bytes allocated)
	00000011	
	01000100	
End of module	00001110	End module
	00000000	0 (error severity)

A table summarizing control byte codes for object language load items is given below.

Object Code Control Byte	Type of Load Item
0 0 0 0 0 0 0 0	Padding
0 0 0 0 0 0 0 1	Add constant
0 0 0 0 0 0 1 0	Expression end
0 0 0 0 0 0 1 1	Declare external definition name
0 0 0 0 0 1 0 0	Origin
0 0 0 0 0 1 0 1	Declare primary reference name
0 0 0 0 0 1 1 0	Declare secondary reference name
0 0 0 0 0 1 1 1	Define field
0 0 0 0 1 0 0 0	Define forward reference
0 0 0 0 1 0 0 1	Declare dummy section
0 0 0 0 1 0 1 0	Define external definition
0 0 0 0 1 0 1 1	Declare standard control section
0 0 0 0 1 1 0 0	Declare nonstandard control section
0 0 0 0 1 1 0 1	Define start
0 0 0 0 1 1 1 0	Module end
0 0 0 0 1 1 1 1	Repeat load
0 0 0 1 0 0 0 0	Define forward reference and hold
0 0 1 0 0 0 R R	Add value of declaration
0 0 1 0 0 1 R R	Add value of forward reference
0 0 1 0 1 0 R R	Subtract value of declaration
0 0 1 0 1 1 R R	Subtract value of forward reference
0 0 1 1 0 0 R R	Change expression resolution
0 0 1 1 0 1 R R	Add absolute section
0 0 1 1 1 0 R R	Subtract absolute section
0 1 0 0 N N N N	Load absolute
0 1 0 1 Q C R R	Load relocatable (long form)
1 C D D D D D D	Load relocatable (short form)

APPENDIX B. FILE ORGANIZATION

INTRODUCTION

A file is an organized collection of information that may only be created, modified or deleted through the Monitor system. A file has one base name but may have other names synonymous with it.

Information is retrieved from a file by specifying the file name, password, account, and the desired record within the file.

The Monitor maintains a master directory of those accounts (users) that maintain files. Each entry in the master directory points to the user's master index of files. Each entry contains the access code and, in general, all significant attributes of the file.

FILE CONTROL

Each file has an independent means of controlling the way in which it is used.

Information is not made available to users who are no authorized to obtain it. With each file, a password and information as to which user may read and/or update the file is recorded. Protection from unauthorized disclosure is attained by checking the information carried with the file against the information supplied by the user.

Changes to the file are allowed or disallowed based on the user's password and account. No accidental changes can occur.

Files may be shared simultaneously or a user may attain exclusive use. In addition, individual records within a file may be designated for exclusive use, though this does not preclude the remainder of the file from being shared.

A user cannot create a file by using an account number other than his own.

FILE ORGANIZATION

Keyed files are those in which each record has an identifying key associated with it. A key consists of a character string, the first byte of which states the number of characters in the string.

1. Keyed Files on Disc

   As the file is being created, a master index is also created with an entry for each keyed record in the file. The entry contains such information as the disc address of the record, size of the record, and position of the record within the blocking buffer.

   The records are packed into blocking buffers with the last portion of the last record extending into another buffer as necessary. If the record is large, it is written directly from the user's area instead of being packed into a buffer. Keyed files on disc may be accessed by direct or sequential access.

2. Keyed Files on Tape

   Tape files are created in a manner similar to disc files. The key associated with a record is stored at the front of the record. Records are packed into 512-word blocks and written to tape as the block is filled. Large records are written directly from the user's area. The last part of the last record may extend into the next block if necessary. No master index is maintained.

Consecutive files are files whose records are organized in a consecutive manner; i.e., there are no identifying keys associated with the records. The records may only be accessed sequentially.

1. Consecutive Files on Disc

   Records are packed into blocking buffers and written on disc. A master index is maintained containing an entry for each granule used. The information in the index is similar to that for keyed files.

2. Consecutive Files on Tape

   Records are packed into 512 word blocks and written to tape. No master index is maintained.

FILE ACCESS

Sequential Access

Sequential access may be used when accessing records with keyed or consecutive organization.

1. Output Files

   Records of files in the output mode may only be written – reading is not allowed.

   If the file has been declared a keyed file, a key must be given with each write operation and this key must be a new key (i.e., it must not have been used before). If the key has already been used, no information is written and an X'16' abnormal return is executed. The keys must be given in a sorted order. For example, if the user writes records with keys A, C, D, respectively, and then writes a record with key B, the record will not be written and an X'18' error return will be executed.

   The PRECORD FWD (position record forward or FORESPACE) and PRECORD REV (position record backwards or BACKSPACE) operations are allowed on both keyed and consecutive files. A BOF is given when the beginning-of-file is reached and an EOF is given when the end-of-file is reached. On tape files, BACKSPACE and FORESPACE will cause the tape to be positioned as specified.

   On disc files, the pointer to the current entry in the master index is decremented or incremented. A WRITE operation following BACKSPACE or FORESPACE causes all forward records to be deleted.

A WRITE with a 0 count will cause a null record to be written; on keyed files, the key is saved but no record is written.

When closing the file, the SAVE option must be specified in the CLOSE statement and in the OPEN statement if the file is to be saved. If the file already exists, and generation numbers are not used, the new file replaces the old one. If generation numbers are used, the new file becomes the latest generation.

2. Scratch Files

The same rules that apply to output files also apply to scratch files, except that reading is allowed. Reading may be directional — either forward or reverse. A READ with REV implies that the record preceding the current position is to be read. If no direction is specified, FWD is assumed.

When reading a keyed file, a key may or may not be specified. If a key is specified, a search is made for the specified key. The FWD, REV options are ignored when a key is specified. If a key is not specified, READ FWD implies that the next record in sequence is to be read. READ with REV implies that the record immediately preceding the current record is to be read.

Reading a consecutive file is the same as reading a keyed file without specifying a key.

A WRITE with NEWKEY deletes all forward information.

3. Input Files

Records may only be read — writing is not allowed.

The READ function is the same as that for scratch files.

BACKSPACE and FORESPACE operations are allowed. On a keyed or consecutive tape file, BACKSPACE and FORESPACE will cause the tape to be positioned as specified.

4. Update Files

The READ function is the same as for scratch files. Positioning operations are allowed.

The WRITE function may or may not have a key specified. If a key is not specified, the WRITE function must have been preceded by a READ. If it is, the record just read is updated; if not, an X'15' abnormal code is signaled.

New records may be added to the file. On a keyed file, the NEWKEY option must be specified, and a search of the keys will be made to locate the proper place to merge the new key. If the key already exists, an X'16' abnormal return is executed.

If the file is consecutive, new records may be added by positioning to the end of file and giving WRITE operations.

The DELETE function may be used. If a key is specified, a search of the directory is made to find the specified key. The record is then deleted. If a key is not specified, the DELETE operation must have been preceded by a READ. Then the key just read will be deleted.

On labeled tapes, a file may be opened in the update mode. However, the first write operation will cause all forward information to be deleted.

Direct Access

Direct access may be used only on files with keyed organization on disc.

1. Output Files

When a WRITE is given, a key must be specified. The keys do not need to be given in a sorted order. They will be ordered as they are stored on disc.

Unlike sequential output files, the last WRITE does not cause forward information to be deleted.

Reading is not allowed.

2. Scratch Files

As for output files, a key must be specified on each WRITE. The keyed record is merged into the file.

A READ may or may not specify a key. If a key is specified, a search is made of the file until the key is found and then the record is read. If the key is not found, an error return is executed.

If a key is not specified, the next sequential record is obtained.

The FWD and REV options apply on READ operations not specifying a key. If a key is specified, these options are ignored. BACKSPACE and FORESPACE operations are performed in the same way as for sequential output files. A WRITE with NEWKEY does not cause forward information to be deleted. A READ after WRITE returns an X'06' EOF.

3. Input Files

This is the same as for sequential input files.

4. Update Files

This is the same as for sequential update files. READ after WRITE returns an X'06' EOF.

RECORD BLOCKING

Records may be written to tape or disc in the blocked or unblocked mode.

Keyed Files and Consecutive Files on Disc (blocked)

The system will automatically block records to provide more efficient use of disc space. The user has no knowledge of this blocking and, when reading, will receive the appropriate record within the block and not the entire block. When updating, the user may rewrite a record in a size larger or smaller than the original record size. If necessary, the Monitor will allocate additional disc space to accommodate the larger size.

Keyed Files and Consecutive Files on Tape (blocked)

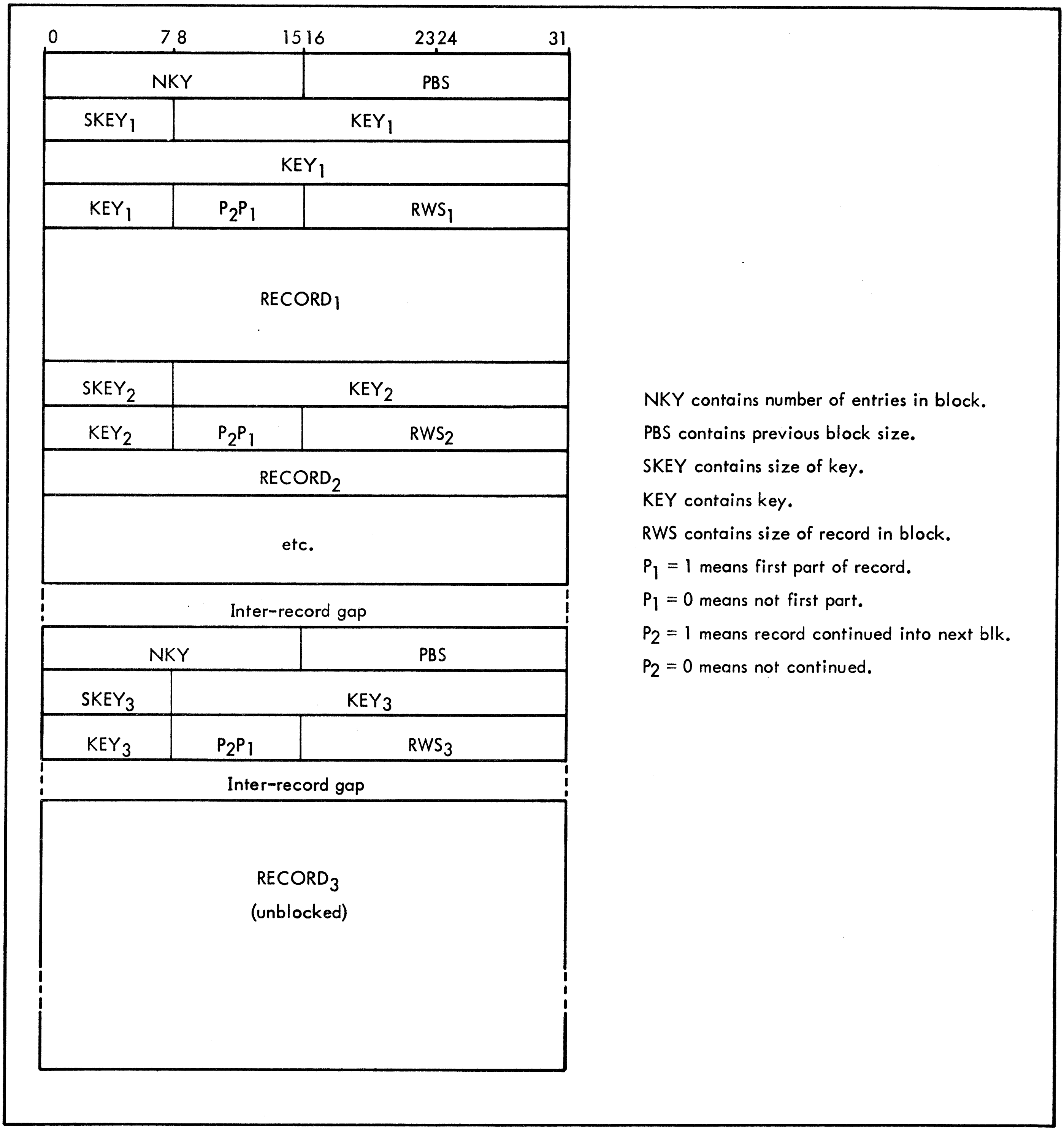
(See Figure B-1.)

Blocking will be performed similarly to blocking disc records.

To perform BACKSPACE or FORESPACE operations, the correct tape positioning will be done by reading each block and determining the number of records within the block.

EXCLUSIVE USE OF RECORDS AND FILES

A user is given the ability to prevent other users from accessing a record and/or file until desired operations are complete.



NKY contains number of entries in block.

PBS contains previous block size.

SKEY contains size of key.

KEY contains key.

RWS contains size of record in block.

P_1 = 1 means first part of record.

P_1 = 0 means not first part.

P_2 = 1 means record continued into next blk.

P_2 = 0 means not continued.

Figure B-1. Labeled Tape Format for Variable-Length Blocked Records

EXCLUSIVE USE OF TAPE FILES

1. Single-File Tapes

   Once a user has opened a file, no other users may access the file until the original user closes it.

2. Multi-File Tapes

   Once a user has opened a file on a multi-file tape, no other user may access the tape until the original user has closed the file. If the REW option is specified the tape is rewound and a message is typed requesting the operator to dismount the reel. Otherwise, the tape remains at the current position and, if a DCB is opened using that tape, one of two actions occurs:

   a. On input or update, the tape is scanned forward for the desired file.

   b. On output, the tape is positioned to the end of the current file and the new file is written at that position.

EXCLUSIVE USE OF DISC FILES

1. Output Files

   When creating a file, the user has exclusive use of the file until it is closed.

2. Scratch Files

   Exclusive use is implied until the file is closed.

3. Input Files

   Exclusive use is not given.

4. Update Files

   On keyed files using direct access, the user will obtain exclusive use of the record by executing a read operation.

   If, at any time, the user does not want exclusive use when executing a read operation, he may use the NOEXCL option.

MULTIPLE DCB'S WITHIN A USER'S PROGRAM REFERENCING THE SAME FILE

TAPE FILES

1. Single-File Tapes

   Only one DCB may be opened to the file at a time. If an attempt is made to open a tape file that is already referenced by another DCB, an abnormal return will be executed.

2. Multi-File Tapes

   Only one DCB may be opened to the tape at a time. The user may not reference another file on the tape until the previous file is closed. To do so will cause an abnormal return.

DISC FILES

1. Output Files

   The user may have only one open DCB in the output mode referencing an output file (a "new" file). However, if another version of the file (a "current" file) already exists on disc, any number of DCBs in the input mode may read that file. If the new file is closed with the SAVE option, the disc space for the current file is released and the new disc space is saved. For this reason, all DCBs referencing the current file must be closed before the new file is closed; if the DCBs are not closed, an X'51' abnormal return is made. Furthermore, the user must open the new file first.

2. Scratch Files

   See "Output Files", above.

3. Input Files

   Any number of DCBs with the input mode specified may reference the same file.

4. Update Files

   The user may have only one open DCB in the update mode referencing a file.

LABELED TAPES (See Figure B-2)

SENTINELS

The formats of sentinels for labeled tapes are described below. All sentinels begin on a word boundary.

1. :LBL

   This record identifies the reel number of the tape. Reel numbers are four alphanumeric characters in length. Sentinel length: 8 bytes (see Figure B-3).

2. :ACN

   This sentinel identifies the owner of the tape, the expiration date, and the creation date, in that order.

   The account number is 8 alphanumeric characters in length, left-justified and in EBCDIC code (see Figure B-4).

Tape 1

Label sentinel (:LBL)
Identification sentinel (:ACN)
Tape mark
Beginning of file A (:BOF)
User's label
Tape mark
Record 1 of file A
Record 2 of file A
Record 3 of file A
Tape mark
End of volume (:EOV)
Tape mark
End of reel (:EOR)
Tape mark
Tape mark

Tape 2

Label sentinel (:LBL)
Identification sentinel (:ACN)
Tape mark
Beginningoffile A (:BOF)
Tape mark
Record 4 of file A
Tape mark
End of file A (:EOF)
Tape mark
Beginning of file B (:BOF)
Tape mark
Record 1 of file B
Tape mark
End of reel (:EOR)
Tape mark
Tape mark

Figure B-2. General Format of Labeled Tape

:	L	B	L
x	x	x	x

Figure B-3. Label Sentinel

:	A	C	N
a_1	a_2	a_3	a_4
a_5	a_6	a_7	a_8
m_1	m_2	d_1	d_2
ƀ	ƀ	y_1	y_2
m_1	m_2	d_1	d_2
ƀ	ƀ	y_1	y_2
Inter-record gap			
Tape mark record			

Figure B-4. Identification Sentinel

The dates are of the form $m_1m_2d_1d_2$ƀƀy_1y_2, where m_1m_2 is the numerical representation of the month, d_1d_2 the day, ƀƀ are blanks, and y_1y_2 are the last two digits of the year. The digits are in EBCDIC and the blanks must appear.

Sentinel length: 28 bytes followed by a physical end-of-file (tape mark record).

3. :BOF

The beginning-of-file sentinel consists of the file-information record, the user's label (if the user has specified one) and a physical end-of-file. The file information consists of control words and the information itself (see Figures B-5 and B-6). A control word has the following form:

Code	LEI	Length

a. "Code" identifies the type of information following the control word.

The codes are:

01 - file name. The file name may be a maximum of 31 characters. An additional byte is used to state the length of the file name.

03 - password (2 words, left-justified).

05 - READ account numbers.

06 - WRITE account numbers.

Each account number is left-justified, blank-filled, and two words long. The total number of READ and WRITE accounts must not exceed 16. READ account identify those who may have only read access to the file. WRITE accounts identify those who may read and write the file. NONE or ALL are also allowed.

09 - miscellaneous information, such as:

ORG – gives the file organization, which may be keyed or consecutive.

KEYM – specifies the maximum length of the keys. Keys may not be greater than 63 bytes. An additional byte is used to specify the length of the key. On consecutive files, the length of the dummy key is assumed to be two; therefore, KEYM is ignored. On keyed files, if KEYM = 0, the maximum length is assumed to be 11.

VOL – On multi-reel files, this entry specifies the position of this tape in the file. For example, VOL = 2 implies this is the second tape of the multi-reel file. VOL = 1 indicates the beginning of the file. Every file begins with VOL = 1 (including single-reel files).

HDL – This specifies the length of the user's label. If HDL = 0, then no user's label exists and the following record must be a physical end-of-file.

BLK – specifies the type of structure of the record formats. If BLK = 1, then the file is unblocked. If BLK = 0, the file is blocked.

b. LEI is the last-entry indicator; this entry in the control word indicates the end of the file information. The control words, along with the information they define, do not have to be in a particular order, but LEI must equal 0 if the file information entry is not the last one and must equal 1 if the entry is the last one.

c. "Length" specifies the length, in words, of the information associated with a particular entry (i.e., following the code word).

4. :EOF, :EOV, and :EOR

These sentinels are described in Figure B-7, B-8, and B-9. The notation "PBS" represents the value of the previous block size, in bytes.

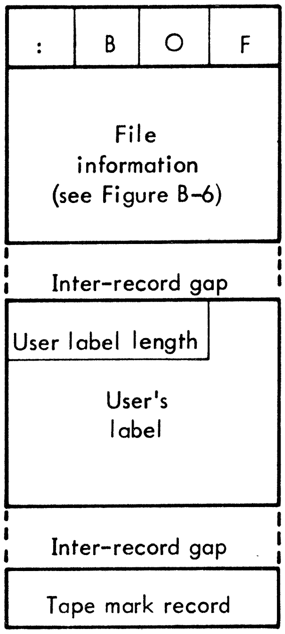


Figure B-5. Beginning-of-File Sentinel

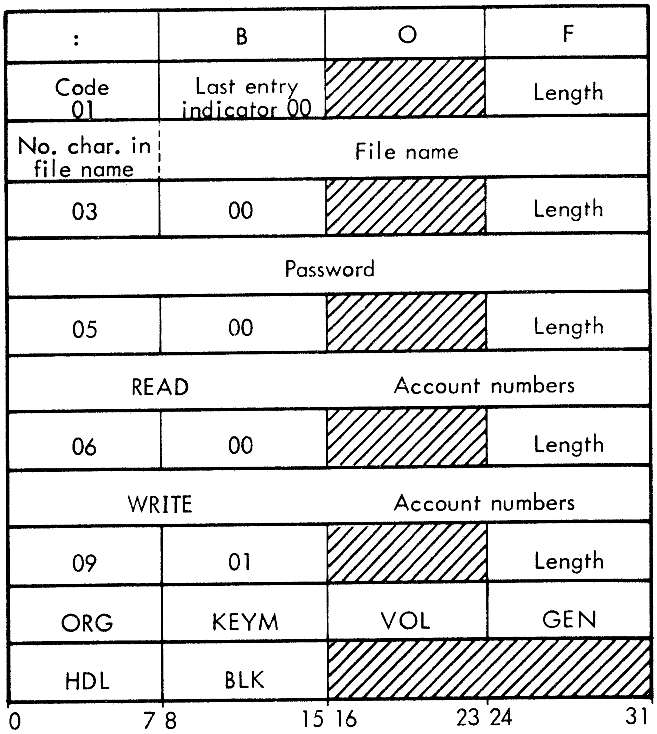


Figure B-6. File Information on Tape

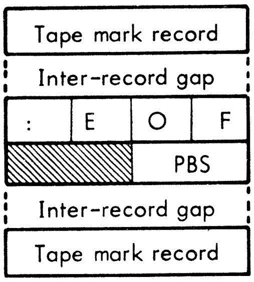


Figure B-7. End-of-File Sentinel

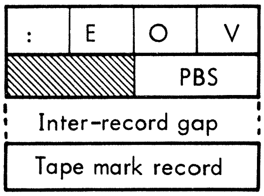


Figure B-8. End-of-Volume Sentinel

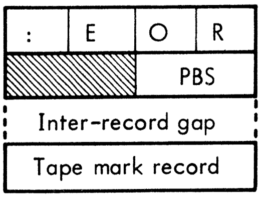


Figure B-9. End-of-Reel Sentinel

APPENDIX C. I/O HANDLERS

INTRODUCTION

I/O handlers are routines selected during System Generation that allow communication between the computer and its peripheral devices.

With the exception of 7TAP (called via MTAP), the handlers are completely independent of each other, and any combination of handlers is operable without the remainder being utilized. Functions common to various handlers are performed by subroutines provided in the Monitor system (handlers themselves provide only for specialized I/O and error recovery).

The handlers can service many devices simultaneously in a reenterable fashion, and are parametrically organized. The specialized handlers available under the Monitor are listed as follows:

KBTIO	(Keyboard Typewriter Handler)
CRDIN	(Card Reader Handler)
CRDOUT	(Unbuffered Card Punch Handler)
PRTOUT	(Buffered Line Printer Handler)
PTAP	(Paper Tape Handler)
MTAP	(Magnetic Tape Handler)
7TAP	(7 Track Magnetic Tape Handler)
DISCIO	(Magnetic Disc Handler)

REGISTERS

The registers used by the various routines are specified in the individual handler descriptions in the latter portion of this section.

The general registers are referred to by the symbols illustrated in Table C-1.

Table C-1. Register References

Register	Symbol	Note	Handler Use
0	R0	1	Not used by I/O handlers.
1	R1	1	
2	R2	1	
3	R3	1	
4	R4	1	
5	R5	3	Required by I/O routines to contain the full I/O address. R5 gets status and device address as a result of an AIO instruction just prior to entering any I/O interrupt service routine.
6	R6	1	Not used by I/O handlers.
7	R7	3	Required by I/O routines to contain the logical DC number (pointer to correct entry in DCT1 through DCT15).
8	SR1	2,3	Contains function code and DCB address.
9	SR2	2,3	Not used by I/O handlers.
10	SR3	2,3	
11	SR4	2,3	Used for linkage address for subroutines.
12	D1	1	Contains the TIO instruction results just prior to entry of a handler or interrupt service routine.
13	D2	1	
14	D3	1	Not used by I/O handlers.
15	D4	1	

Notes: 1. These registers are considered as volatile registers and may be used and destroyed by any routine.

2. System registers.

3. These registers must be preserved and restored if used by any subroutine.

Note: The handlers can use TSTACK for temporary storage (TSTACK is the location of a Stack Pointer Doubleword). All storage used in a temp stack is released before the associated routine exits.

BASIC STRUCTURE

Upon entry, with SR4 as the linkage register, the handler sets up the order codes and interrupt flags in the command pairs in the doubleword location(s) specified by the address in DCT9+R7. The handler sets register D3 to the error return address in case there is an unsuccessful reply to the SIO instruction, and uses register D4 as the link to the SIO instruction routine.

A set of common interface subroutines is called that tests for device ready condition, sends the SIO instruction, and tests for device address recognition and SIO acceptance. If the SIO is not accepted, the error service routine address within the handler is set (for key-in recovery) into DCT7+R7. If the SIO is accepted, DCT7+R7 is set to the address of the interrupt service routine. The subroutine returns to the Monitor or I/O executive via SR4.

Upon reentry and following the I/O termination interrupt, the termination status (from the AIO) in R5 is checked for error conditions. Each handler takes specific action if an error has occurred. The number of recovery tries remaining is checked in cases where automatic error recovery is possible. If more recovery tries remain, the operation is repeated by reentering the first part of the handler via SR4, and then exiting to IOCONT. If the number of tries has been exhausted, TYERROR is entered and the return address in DCT7+R7 is set for operator key-in recovery.

If no error has occurred, the number of bytes transmitted is stored in DCT7+R7. Output handlers exit to OCOMP. Input handlers check for lost data and report the condition to the user's DCB, via the ABNCOND routine. A check is made for end-of-file and change of mode. If end-of-file is detected, an EOF condition is reported to the user's DCB via the ABNCOND routine. If a change-of-mode control record is read, the mode indicator in DCT1+R7 is changed, and the next record is then read in the new mode.

ENTRY AND EXIT

All handlers are entered from a common point in the I/O executive. Register SR4 contains the return address and each handler returns via SR4 after setting its I/O termination interrupt service routine address (in DCT7).

Any handler will be reentered at its I/O termination interrupt service routine, where termination status may be checked and an appropriate response made. Several types of error and successful completion are possible.

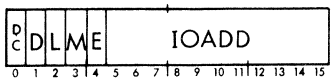
If the operator specifies that an operation is to be retried (as the result of an error status response notification), the appropriate subroutine (whose address is in the interrupt return table) is executed and the subroutine returns via SR4.

DEVICE TABLES SET/USED

The device tables are constructed parallel to one another. Each has as many entries as there are devices in the system. Therefore, the 'i'th entry in each table corresponds to the same device. The '0'th entry in each table is null. Agiven table may be a byte, halfword, or word resolution. Information in a particular table may therefore be accessed by using the table name as modified by the device pointer (entry number) carried in index register R7. The various tables are as follows:

Used and set by executive and by handlers:

DCT1

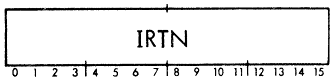


DCT1 (halfword table)

DC	= 1	Means DC busy
D	= 1	Means device busy
L	= 1	Means next device on same DC
M	= 1	Means binary, 0 - EBCDIC (used and set by handlers)
E	= 1	Means an error is pending
IOADD		Contains full I/O address of device

Set by handlers and used by executive:

DCT7

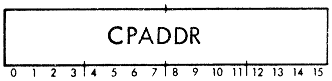


DCT7 (halfword table)

IRTN contains I/O service routine interrupt address set by first part of handler or actual record size (in bytes) set by last part of handler

Set by executive for use by handlers:

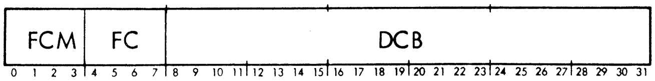
DCT9



DCT9 (halfword table)

CPADDR contains address of doublewords required to form I/O commands

DCT10



DCT10 (word table)

FCM contains function code modifiers

FC contains function code

DCB contains DCB address (point to the parameter list utilized by the executive to determine the nature of the I/O request)

DCT13

SKIPC
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

DCT13 (halfword table)

SKIPC contains magnetic tape skip count

HANDLER AND I/O EXECUTIVE INTERFACE

The I/O executive sets a pointer (R7) to the proper entry number in the DCT device tables before entering the handler for that device (i.e., the device corresponding to R7). It sets register SR1 to the contents of DCT10+R7 (see Table C-2); sets the byte address of the buffer and byte count for transfer in the first two words of the command list (saving the doubleword address of the list in DCT9+R7); and sets the I/O device address from DCT1+R7 into register R5. The function codes passed via DCT10+R7 are given in Table C-2a.

Table C-2a. Function Codes Passed via DCT10+R7

Function Code (Bits 4-7)	Function	Applicable Devices
0	Read	Card reader, keyboard, paper tape leader, magnetic tape
1	Read Direct	Paper tape reader, card reader
2	Read Binary	Magnetic tape
3	Read Direct Binary	Magnetic tape, card reader, paper tape
4	Write Direct	Paper tape
5	Write BCD	Card punch, typewriter, paper tape punch, line printer, magnetic tape
6	Write Binary	Card punch, paper tape punch, magnetic tape
7	Write Direct Binary	Paper tape
A	Skip Records Forward	Magnetic tape
B	Skip Records Reverse	Magnetic tape
C	Skip File Forward	Magnetic tape
D	Skip File Reverse	Magnetic tape
E	Rewind	Magnetic tape
F	Write EOF	Magnetic tape

Table C-2b. Function Code Modifiers

Bit	Meaning[†]
0	Not used
1	Set = 0 means forward
	Set = 1 means reverse
2	Set = 0 means unpacked mode
	Set = 1 means packed mode
3	Set = 0 means no FORTRAN BCD conversion
	Set = 1 means FORTRAN BCD conversion

Note: The read direct and write direct functions signify that Monitor formatting conventions are to be ignored. Most handlers do not use formatting in any case, and the function codes for direct reading or writing are treated in the same fashion as the normal function codes. The card reader and paper tape handlers do react to the direct function codes.

The names and functions of I/O handler interface routines are given in Table C-3. The interface routines utilized by the various handlers are specified in the individual handler descriptions.

Table C-3. Names/Functions of Handler Interface Routines

Names	Function
COMSIOCL	Sends SIO, checks acceptance and empty status.
SENDSIO SENDSIO 1	Sends SIO and checks acceptance.
EMPTYCK	Checks empty status after SIO accepted.
MODBIT	Sets a bit in the order code of the first command.
STORE4WD	Stores 4 words from R2, R3, D1, D2.
FUNCTION	Sets order code to write, then checks type of function.
REGETWD	Sets R2, R3, D1, D2 with the first four command words, sets R1 to the doubleword address of the command list, and sets R0 to the first byte of the image in the I/O buffer.
NEWLINE	Checks for image of NL in R0 and does some housekeeping (for KBT and paper tape reads in BCD mode).

[†] Modifier bits are ignored if the function code (FC) has a value of A_{16} or greater.

Table C-3. Names/Functions of Handler Interface Routines (cont.)

Names	Function
CKERRLDT CKERRLD2	Checks for transmission errors.
NOERR	Sets numbers of bytes read and checks for lost data.
CKLODATA	Reports lost data, if any.
CKBCDBIN	Checks record image for ! or !EOD, !BIN or !BCD and reports presence as applicable.
BCDCHANG	Changes mode bit in DCT1+R7 to BCD.
BINCHANG	Changes mode bit in DCT1+R7 to BIN.
EODNOTE	Reports EOD condition with flag via ABNCOND.

I/O HANDLER DESCRIPTIONS

A description of the individual I/O handlers used under the Monitor system is given in the following pages of this appendix.

KBTIO (Keyboard Typewriter Handler)

PURPOSE

Specific keyboard typewriter action.

INITIAL CONDITIONS

DCT	tables set
R5	set with device address
R7	set as pointer to pertinent DCT entry for this device
SR1	set with function code
SR4	set as linkage return address

CALLING SEQUENCE

BAL, SR4 KBTIO

SUBROUTINES AND PROCEDURES USED

System	Handler Interface
ABNCOND	REGETWD
IOCONT	FUNCTION
ICOMP	STORE4WD
	SENDSIO
	NEWLINE
	SENDSIO1
	CKBCDBIN

REGISTERS USED

R0 is set with the doubleword address of the command list (DCT9+R7).

R2, R3 and D1, D2 are used for intermediate storage of actual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction and to check error conditions after interrupt, and also used for temporary storage for words 5 and 6 of the command list.

RESULTS

A specific I/O operation is transacted.

DESCRIPTION

The following commands are applicable:

Read – Read bytes and edit for special character until an NL (15_{16}) byte is read or the byte count is exhausted, whichever occurs first. If the count is exhausted first, output an NL (15_{16}) byte. Then, if !EOD was read, report EOD via ABNCOND. If a rub-out byte (FF_{16}) is encountered, do not store the rub-out byte, but store the next byte over the preceding byte. If an EOM (08) byte is received, output an NL (15_{16}) byte and reinitiate the read byte count and beginning address.

Some of the code for this handler is shared with the Paper Tape BCD Read handler and is located in the set of handler interface routines mentioned above.

Write BCD Write Binary	Type the specified buffer.

CRDIN

PURPOSE

Specific card reader action

INITIAL CONDITIONS

DCT	tables set
R5	set with device address
R7	set as pointer to pertinent DCT entry for this device
SR1	set with function code
SR4	set as linkage return address

CALLING SEQUENCE

BAL, SR4 CRDIN

SUBROUTINES USED

System	Handler Interface
IOCONT	MODBIT
ICOMP	COMSIOCL
TYERROR	CKERRLDT
TYINTLK	CKBCDBIN
TYBADIO	
TYEMPTY	
ABNCOND	

REGISTERS USED

R0 is set with the doubleword address of the command list (DCT9+R7).

R2, R3 and D1, D2 are used for intermediate storage of actual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction and to check error conditions after interrupt.

RESULTS

A card has been read.

DESCRIPTION

The following functions are available:

Read – Read the next card according to the current mode of the device. If the SIO instruction is not accepted, report it and reenter the handler. If data is lost, report LDT via ABNCOND. If !BCD is encountered, set device mode to BCD and read. If !BIN is encountered, set device mode to binary and read. If !EOD is encountered, report EOD via ABNCOND and exit.

Read Direct – Read the next card according to the current mode of the device. If data is lost, report LDT via ABNCOND.

The handler is reentered at the beginning (after any error is reoprted) and continuation is specified.

CRDOUT (Card Punch Handler, Unbuffered)

PURPOSE

Specific card punch action.

INITIAL CONDITIONS

DCT tables set

R5 set with device address

R7 set as pointer to pertinent DCT entry for this device

SR1 set with function code

SR4 set as linkage return address

CALLING SEQUENCE

BAL, SR4 CRDOUT

SUBROUTINES USED

System	Handler Interface
IOCONT	REGETWD
OCOMP	COMSIOCL
TYERROR	CKERRLD2
TYBABADIO	
TYEMPTY	
CHKERC	

REGISTERS USED

R0 is set with the doubleword address of the command list (DCT9+R7).

R2, R3 and D1, D2 are used for intermediate storage of actual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction and to check error conditions after interrupt, and also used for temporary storage for words 5 and 6 of the command list.

RESULTS

A card has been punched.

DESCRIPTION

The following functions are available:

Write BCD – Punch the specified image (up to 80 bytes) in EBCDIC.

Write Binary – Punch the specified image (up to 120 bytes) in Binary.

If errors are detected, card in error plus the following card are directed to the error stacker and both cards are repunched.

The handler is reentered at a point just after the initial housekeeping code if there is continuation following an unaccepted SIO, and at a buffer switching routine following a transmission error continuation.

PRTOUT (Line Printer Handler, Buffered)

PURPOSE

Specific line printer action.

INITIAL CONDITIONS

DCT	tables set
R5	set with device address
R7	set as pointer to pertinent DCT entry for this device
SR1	set with function code
SR4	set as linkage return address

CALLING SEQUENCE

```
BAL, SR4      PRTOUT
```

SUBROUTINES AND PROCEDURES USED

System	Handler Interface
IOCONT	MODBIT
OCOMP	COMSIOCL
TYERROR	CKERRLDT
TYINTLK	
TYBADIO	
TYEMPTY	
CHKERC	

REGISTERS USED

R0 is set with the doubleword address of the command list (DCT9+R7).

R2, R3 and D1, D2 are used for intermediate storage of actual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction and to check error conditions after interrupt, and are also used for temporary storage for words 5 and 6 of the command list.

RESULTS

A line has been printed. Format control has taken place if requested.

DESCRIPTION

The following commands are applicable:

Write BCD – Print the specified buffer as text.

Write with format – Print the specified buffer using the first byte as a format control character.

The handler is reentered at the beginning if any error is reported and if continuation is specified.

PTAP (Paper Tape Handler)

PURPOSE

Specific paper tape action.

INITIAL CONDITIONS

DCT	tables set
R5	set with device address
R7	set as pointer to pertinent DCT entry for this device
SR1	set with function code
SR4	set as linkage return address

CALLING SEQUENCE

```
BAL, SR4      PTAP
```

SUBROUTINES AND PROCEDURES USED

System	Handler Interface
IOCONT	REGETWD
IOCOMP	FUNCTION
OCOMP	STORE4WD
TYERROR	COMSIOCL
TYINTLK	CKERRLDT
TYBADIO	BINCHANG
TYEMPTY	BCDCHANG
	CKLODATA
	NEWLINE
	CKBCDBIN

REGISTERS USED

R0 set with the doubleword address of the command list (DCT9+R7).

R2, R3 and D1, D2 are used for intermediate storage ofactual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction, to check error conditions after interrupt, and used for temporary storage for words 5, 6 of command list.

RESULTS

Input or output has been done. Abnormal code may be set.

DESCRIPTION

The following commands are applicable:

Read – Read first nonblank byte. If the byte is 11_{16}, this is a binary record. Read the next two characters and interpret as record size. Then read specified number of bytes. If the record size is greater than the requested size, report LDT via ABNCOND and bypass the remainder of record. Set the device mode to binary.

If the byte is not 11_{16}, this is an EBCDIC record. Read bytes and transmit to the user buffer until record size is reached, or an NL(15_{16}) character or blank frame (00) is encountered. If the record size is reached, set LDT via ABNCOND and bypass the remainder of the record. If FF_{16} is encountered, do not transmit to the user's buffer. If EOM (08) is encountered, reset the byte address to the beginning of the user's buffer. Set the device mode to EBCDIC.

If !EOD is the final image, report EOD via ABNCOND.

The following abnormal conditions apply:

EOD – Any !EOD read as first four bytes.

Mode Change – Mode is set according to mode of record (see above).

LDT (lost data) – BCD or Binary record longer than requested length.

EBCDIC records are made up of one or more characters followed by a carriage return. Input is in the single character mode, and the leader is ignored. The characters are checked as they are read. A carriage return signifies the end of the record, regardless of the byte count. It the byte count is exceeded, the remainder of the record is ignored. An EOM character means to ignore the entire record to this point. Parts of the code for editing EBCDIC records are found in the handler interface routines and paper tape or typewriter input handlers. The rub-out character on paper tape is ignored and, on the typewriter, causes the previous character to be ignored (in addition to the rub-out character). Carriage returns are not transmitted to the user.

Binary records are made up of a one-byte header (11_{16}) followed by a two-byte record count and data bytes.

Read Direct – Read specified number of bytes into the user's buffer without editing. Report EOD if !EOD is read.

Write BCD – Punch record as specified, followed by two blank (00) frames.

Write Binary – Punch 3 bytes of control information (11_{16}), followed by a two-byte record size, followed by the record as specified, followed by two blank frames (00).

Write BCD Direct
Write Binary Direct } Punch record as specified with no formatting.

In the case of any read or punch errors, no automatic recovery attempts are made. The appropriate message to the operator is typed in all cases.

The handler is reentered at a housekeeping routine and then is restarted at the beginning after any error is reported and a continuation is specified.

MTAP (Magnetic Tape Handler)

PURPOSE

Specific magnetic tape action.

INITIAL CONDITIONS

DCT	tables set
R5	set with device address
R7	set as pointer to pertinent DCT entry for this device
SR1	set with function code
SR4	set as linkage return address

CALLING SEQUENCE

```
BAL,SR4     MTAP
```

SUBROUTINES AND PROCEDURES USED

System	Handler Interface
IOCONT	REGETWD
ICOMP	STORE4WD
OCOMP	COMSIOCL
TYERROR	NOERR
TYINTLK	
TYBADIO	
CHKERC	
ABNCOND	

REGISTERS USED

R0 is set with the doubleword address of the command list (DCT9+R7).

R2,R3 and D1,D2 are used for intermediate storage of actual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction and to check error conditions after interrupt, and used for temporary storage for words 5 and 6 of the command list.

RESULTS

Specified action has been taken.

DESCRIPTION

The following commands are applicable:

Read forward – Read the next record in the forward direction.

Read reverse – Read the next record in the reverse direction.

Write – Write the specified record.

Skip records forward – Skip n records in the forward direction.

Skip records reverse – Skip n records in the reverse direction.

Skip file reverse – Skip forward past the EOF mark.

Skip file reverse – Skip past the EOF mark in the reverse direction.

Rewind – Rewind the tape, report the load point.

Write EOF –Write an EOF mark.

The following abnormal conditions apply:

EOF – Read forward, Read reverse, Skip records forward, Skip records reverse.

LDT (Lost Data) – Read forward, Read reverse.

EOT (End of Tape) – Write, Write EOF, Read forward, Skip forward.

BOT (Beginning of Tape) – Any reverse tape function.

Recovery from errors is as follows:

Read error – Backspace and reread until the error count is exhausted. Then call ERRCOND.

Write error – Backspace, erase, and rewrite until the error count is exhausted. Then call ERRCOND. No retries are allowed if write protect violation.

The rewind operation must request a device interrupt upon completion. The device busy flag must also be set.

The following commands are applicable:

Read – Read starting at address specified.

Write – Write starting at address specified.

Write Direct – Write followed by Check Write.

No abnormal conditions are generated by the handler. Errors are retried as many times as possible. If no recovery is possible, ERRCOND is called. If the address is not accepted by the disc unit, TYBADIO is called.

7TAP (7 Track Magnetic Tape)

PURPOSE

Specific 7 track magnetic tape action.

INITIAL CONDITION

(see MTAP)

CALLING SEQUENCE

B 7TAP1

B 7TAP2

SUBROUTINES AND PROCEDURES USED

MTAP1

MTAP2

REGISTERS USED

(see MTAP)

RESULTS

When entered at 7TAP1, the handler first checks if FBCD invert was requested on a write, and edits the buffer as required. The order code is then set up for Read Binary, Read Packed Binary, Read BCD, Write Binary, Write Packed Binary, or Write BCD, as requested. If Read Reverse is requested, a command chain is set up (backspace, read, backspace) to simulate the Read Reverse option on 9 track tape. Exit is then made to MTAP1.

When entered at 7TAP2, the handler does the required editing of the buffer if FBCD was requested on a Read Operation. Exit is then made to MTAP2.

DISCIO (Magnetic Disc Handler)

PURPOSE

Specific disc action.

INITIAL CONDITIONS

DCT	tables set
R5	set with device address
R7	set as pointer to pertinent DCT entry for this device
SR1	set with function code
SR4	set as linkage return address
DCT13+R7	set to disc address

CALLING SEQUENCE

BAL, SR4 DISCIO

SUBROUTINES AND PROCEDURES USED

System	Handler Interface
IOCONT	REGETWD
ICOMP	STORE4WD
OCOMP	COMSIOCL
DKERROR	CKERRLDT
TYINTLK	
TYBADIO	
CHKERC	
ABNCOND	

REGISTERS USED

R0 set with the doubleword address of the command list (DCT9+R7).

R2, R3 and D1, D2 used for intermediate storage of actual command pairs.

D3 and D4 are used for linkage and error address for handler interface subroutines to send the SIO instruction and to check error conditions after interrupt, and used for temporary storage for words 5 and 6 of the command list.

RESULTS

Specified action has been taken.

DESCRIPTION

The following commands are applicable:

Read – Read starting at address specified.

Write – Write starting at address specified.

Write direct – Write followed by Check Write.

No abnormal conditions are generated by the handler. Errors are retried as many times as possible. If no recovery is possible, ERRCOND is called. If the address is not accepted by the disc unit, TYBADIO is called.

APPENDIX D. SENSE SWITCH SIMULATION

The Monitor provides the capability of initializing as many as six pseudo sense switches, by means of the SWITCH control command (see Chapter 2 of this manual) and also provides for setting and resetting them by means of SWITCH unsolicited key-ins (see Chapter 3 of this manual).

Three library routines are also provided to allow processors and user programs to set, reset, and test specified pseudo sense switches. The entire sense switch simulation is based on the use of a pseudo sense switch register contained in a Task Control Block (TCB) established and maintained by the Monitor. The first two words of the TCB comprise a Stack Pointer Doubleword (SPD), and the subsequent words contain additional information used by the Monitor to control the current task. When the Monitor transfers control to a user's program (or a processor), it places the word address of the TCB in general register 0.

When a user's program calls any of the sense switch library routines, general register 0 must contain the word address of the TCB. General register 6 is used for passing the number of the specified sense switch. The link register is general register 11, and general registers 6-10 are volatile (not preserved by the library routine).

TEST-SENSE-SWITCH ROUTINE

The linkage

```
    BAL,11      L:TSS
```

causes the sense switch specified in general register 6 to be tested. If the switch is set, the condition codes are all set (to 1); otherwise, the condition codes are set to 0.

SET-SENSE-SWITCH ROUTINE

The linkage

```
    BAL,11      L:SSS
```

causes the sense switch specified in general register 6 to be set. If the number of the specified switch is not within the range 1-6, the routine will ignore the request.

RESET-SENSE-SWITCH ROUTINE

The linkage

```
    BAL,11      L:RSS
```

causes the sense switch specified in general register 6 to be reset. If the number of the specified switch is not within the range 1-6, the routine will ignore the request.

APPENDIX E. BLOCKING/DEBLOCKING

The Blocking/Deblocking routines will operate under the Monitor and are designed to supplement the Monitor for a given set of special operations on sequential files.

There are three parameters that the user must provide in order to use these routines. They are: block size (RSZ), location of the user's buffer (BUF), and logical record size (LRS). The blocking buffer is used by these library routines and must be at least as large as the block size.

The block size (maximum buffer size) and buffer location (BUF) may be provided by the user when opening a file or may be provided at compile or assembly time in the DCB (see "M:DCB" and "M:OPEN", Chapter 5).

The third parameter, maximum logical record size (LRS), is provided by the user in a parameter list which he provides when calling the routines.

OUTPUT FILES

Logical output records will be packed until either the buffer is full or until the Write Output Buffer routine is performed. The user may obtain control on abnormal or error conditions by specifying an abnormal or error return in the parameter list.

WRITE LOGICAL RECORD

The linkage

```
    BAL,11      L:WLR
```

locates a buffer area large enough to contain the logical record and then moves the record from the user's buffer to that buffer. Whenever insufficient area exists for the next logical record, the records previously blocked will be written. It is assumed that general register 6 contains the location (CLIST) of the parameter list associated with the output request.

CLIST has the following format:

CLIST	LRS	CLRC
PLIST		

The routines defined in this appendix are library routines and, as such, may destroy registers 6-11. Register 6 is used to pass the location (CLIST) of a parameter list and register 11 is used as the link register. The first two bytes of the first word of the list contain the number of bytes (LRS) in the logical record. The next two bytes are assumed to be zero initially and are used by the routines as a current logical record count (CLRC). The remainder of the parameter list (PLIST) is identical to that supported by the Monitor on an I/O READ or WRITE operation (see Chapter 5 of this manual).

WRITE OUTPUT BUFFER

The linkage

```
    BAL, 11        L:WOB
```

causes any unrecorded logical records contained in the buffer to be written. Only the actual records previously blocked will be written. This provides the means for writing variable length physical records. It must also be used before writing an end-of-file, rewinding, closing a file, etc.

It is assumed that general register 6 contains the location (CLIST) of the parameter list associated with the output request.

INPUT FILES

Records input will be read until an abnormal or an error condition is encountered or until a Close Current Input Buffer call is made. The user may obtain control on abnormal or error conditions by specifying an abnormal or error return in the parameter list.

READ LOGICAL RECORD

The linkage

```
    BAL, 11        L:RLR
```

locates the next logical record in the buffer. Only the number of bytes requested will be moved to the user's area. The logical record size requested may never be larger than the logical record size (LRS) provided in CLIST. If no record size (SIZE) is provided in the PLIST, then the logical record size (LRS) provided in CLIST is used. It is assumed that general register 6 contains the location (CLIST) of the parameter list associated with this I/O operation. L:RLR automatically generates a call to M:READ.

CLOSE CURRENT INPUT BUFFER

The linkage

```
    BAL, 11        L:CCIB
```

causes the reading of the next physical record. Any remaining logical records contained in the current buffer will be lost. It is assumed that general register 6 contains the location (CLIST) of the parameter list associated with the input request.

APPENDIX F. DATA FORMATS ON EXTERNAL MEDIA

On external media (magnetic tape, punched cards, and paper tape) it is frequently desirable for an operating system to intersperse certain control information with user data, to maintain system control, device independence (to user), etc. On the other hand, users occasionally desire to control a specific device entirely, as if they were doing the I/O themselves.

These requirements give rise to the need for several formats for external media.

1. Labeled – applies to system maintained files (magnetic tape or disc). A labeled file "appears" to the user as a set of records comprising his data set bounded by a beginning-of-file and end-of-file. Labeled files assume formatted data.
2. Formatted – applies to external media and specifies that I/O records are formatted and/or interpreted by the Monitor. Exact actions are listed below.
3. Direct – applies to external media but specifies that no Monitor formatting of user data is done. End-of-data marks are detected upon reading. The user's I/O request is performed exactly as if he had control of the device.

FORMATTED DATA RECORDS

Formatted data records assume the following forms:

1. Punched Cards – Each record is represented on one card. When the mode is changed (between two records), a mode control card is interjected (!BCD signals that an EBCDIC card follows; !BIN signals that a binary card follows). End-of-data is signaled by an !EOD card.
2. Paper Tape – Each EBCDIC record contains the data followed by an NL (15_{16}) byte or at least two blank (00) frames. Each binary record is made up of a one-byte binary indicator (11_{16} – an invalid EBCDIC code), followed by a two-byte record count, followed by the binary data, followed by at least two blank (00) frames. End-of-data is signaled by an !EOD record.

3. Typewriter – Each record is made up of data of a specified size or terminated by an NL (15_{16}) byte. End-of-data is signaled by an !EOD record.

DIRECT DATA RECORDS

Direct data records assume the following form:

The data records are represented exactly as user specified in all cases. End-of-data is signaled by a physical EOF mark on magnetic tape and by !EOD on cards, paper tape, or typewriter.

The actions resulting from various Monitor I/O requests are as follows:

1. M:WRITE – Write the specified buffer as formatted data. If cards, and mode is changed, output a mode control card before record. Use mode specified in DCB.
2. M:CLOSE (Output File) – Output an end-of-data sentinel and, if paper tape, output 10 inches of leader.
3. M:WEOF – Output an end-of-data sentinel.
4. M:WRITE (Direct) – Output the specified record exactly. If cards, use mode specified in DCB.
5. M:CLOSE (Direct) – No action.
6. M:WEOF (Direct) – Output an end-of-data sentinel.
7. M:CLOSE (Input File) – No action.
8. M:READ – Read the next record, eliminating the format information. Set the mode (DCB) according to the mode of the record. Position to read following record.
9. M:READ (Direct)
   a. Magnetic Tape – Read the next record or specified number of bytes, whichever is smaller. Position to read following record.
   b. Cards – Read the next card in the mode specified by the DCB or the specified number of bytes, whichever is smaller. Position to read the following card.
   c. Paper Tape – Read the specified number of bytes. Position to read the next byte.
   d. Typewriter – Read the specified number of bytes.

ERROR RECOVERY PROCEDURES

1. Magnetic Tape
   a. Input – The record will be read until it is read correctly or the specified number of recovery tries have been made. If it is not read correctly, the operator is notified.
   b. Output – After each error, a gap equal to the length of the record is erased and the operation is retried until it is successful or the specified number of tries have been made. If it is not written correctly, the operator is notified.
2. Cards
   a. Input – The operator is notified.
   b. Output – The action depends on the type of punch. If the device is a multi-stacker punch, the operation is retried the specified number of times. If not successful, the operator is notified. For each recovery attempt, the error card and the following card are diverted to the error stacker and both cards are repunched.
3. Paper Tape

   The operator is notified for all paper tape errors.
4. End-of-File Condition – An end-of-file will be given by the Monitor when a control command is encountered during reading from the control command device (C). Any attempt to read past an end-of-file condition will cause the job to be aborted.

APPENDIX G. COOPERATIVE AND SYMBIONTS

The routines to perform peripheral operations operate concurrently with the jobs being run. The peripheral system is composed of a "cooperative" and a "symbiont" or symbionts. The cooperative is a Monitor routine called as a result of a user's I/O request, whereas a symbiont is a Monitor routine that is initiated either by the action of the cooperative or by operator command from the system console. The cooperative is used to transfer information between the user's program and secondary (disc) storage, and symbionts are used to transfer information between secondary storage and peripheral devices (see Figure G-1).

The symbiont-cooperative system provides for complete buffering between I/O devices and the user's program. Therefore, a user's program never has to wait for an I/O device to complete an action. Also, the current job may be running while the output of the previous job and the job file for the following job are being handled by symbiont operation.

COOPERATIVE

A single cooperative is provided for handling both user input and output files. It is reenterable and can handle any number of device-type files (printer, punch, etc.) per job.

The cooperative will be linked to the user's program via the I/O Supervisor (IOSP). Consequently, the user need not be concerned with modifying his program for operation under either a symbiont or non-symbiont system. In a symbiont-cooperative system, IOSP will link to the I/O peripherals via the cooperative rather than via the I/O Dispatcher.

SYMBIONTS

A symbiont is a small, reenterable routine that controls the action of an I/O device having a lower transfer rate than the computer. Core storage as well as secondary storage may be used to produce a continuous flow of information to or from the peripheral devices. Symbionts may be used to transfer information from one peripheral device to another peripheral device.

Unlike the user's program, which is directed primarily by control commands, symbionts — once initiated — receive all their control from the operator's console. An input symbiont can be initiated only by the console operator, while an output symbiont can by initiated either by the operator or the cooperative. Symbionts are queued for initiation by a Monitor symbiont activation routine, and all communication between symbionts and the operator is through a Monitor table of resident symbionts. All key-ins addressing symbionts are examined, and the current message character is stored in the appropriate table entry. The symbiont table is created at System Generation time.

A symbiont performs only one I/O operation at a time, and is inactive from the time that it initiates a request for I/O until the I/O operation is complete. The symbiont regains control by stipulating an I/O end-action return to itself.

Since symbionts are reenterable, a single symbiont may drive several types of devices. For example, the same symbiont may be used to drive many printers and card punches. The symbiont is given the device table index of the appropriate device, when the symbiont is activated. The index points to corresponding entries in a series of tables providing the device number and type, and the IOP and device controller number. All the peripheral-dependent information is contained in a context buffer. The location of this buffer is made known to the symbiont, via a subroutine linkage register, whenever it is operating on the associated device.

Two symbionts are provided in the Monitor system: one for driving all standard input devices, and one for driving all standard output devices.

SYMBIONT-COOPERATIVE HOUSEKEEPING

Two Monitor subroutines are provided for automatic maintenance of core storage. One is used to release a core buffer from use by the symbionts, and the other is used to obtain a core buffer and its physical location.

If a core buffer is requested by an operating symbiont and none is available, an entry is made in the symbiont core-buffer queue. When one becomes available for symbiont use, control is returned to the requesting symbiont.

As each buffer is emptied, either by reading from or storing into secondary storage, it is released and another is requested by either a symbiont or cooperative. This procedure allows for sharing core buffers where the total number of buffers available may be insufficient for the demand.

Optimum efficiency in buffer utilization is attained if there is one core buffer per device, dedicated to symbiont activity, plus one per I/O type, plus one per symbiont, as well as one context block per I/O type and one context block per device. However, as few as one buffer and one context block could be used per device type, plus one buffer and one context block per symbiont. The total number of core buffers allocated to the system is a System Generation parameter.

The symbionts themselves will operate in core buffers and are self-relocating. When a symbiont is to be initiated, it is read from secondary storage (where it normally resides when not active) into a requested core buffer. After being read into core storage, the symbiont is entered with the appriate peripheral device information. After starting an I/O operation on a peripheral device, with an end-action return specified, the symbiont relinquishes control to the Monitor System which will turn control over to another symbiont or the user's program.

An area of secondary storage is set aside for symbiont files. The size of this area is an installation variable set up at

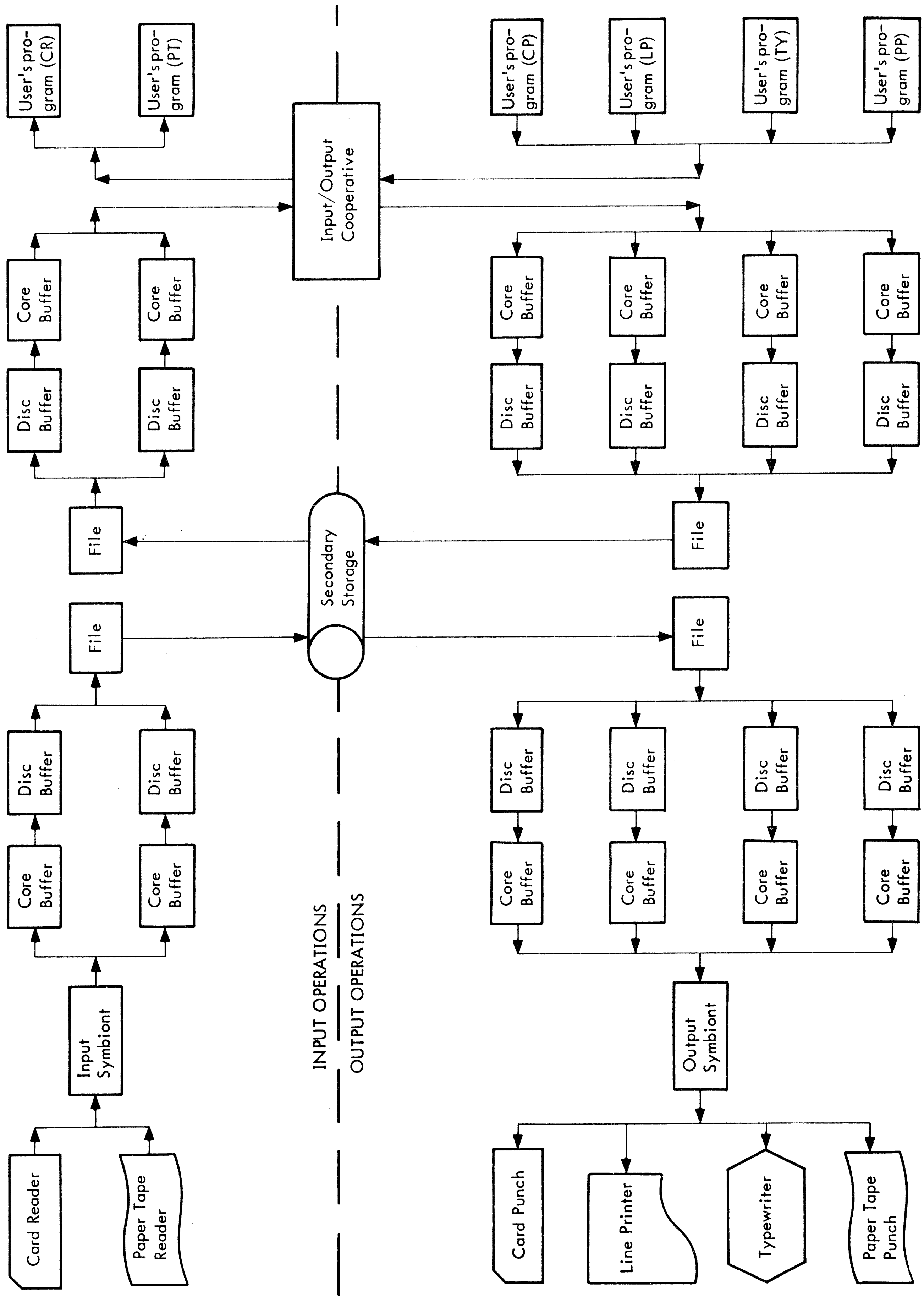


Figure G-1. Information Flow Through Cooperative and Symbionts

System Generation time. A secondary storage allocation table is maintained by the Monitor to indicate which disc areas are available. Two Monitor subroutines are also provided for maintenance of secondary storage. One of these requests storage; the other releases it. If secondary storage is requested and none is available, an entry is made in the symbiont secondary storage queue. As I/O information is processed by the cooperative or symbiont, secondary storage is automatically released by the Monitor. When sufficient storage has been released, control is returned to the requesting symbiont.

Secondary storage holds the files produced by, or committed to, a peripheral device. Each disc block of secondary storage contains the disc address of the next disc block in the file, and a table of job files is maintained by the Monitor. Two Monitor subroutines are provided for file maintenance. One removes a file; the other inserts a new file into the file table. File removal is based on peripheral code, priority number, and system identification. When an input job file is inserted into the file table, the system is taken out of the "idle" state and the Control Card Interpreter (CCI) is activated. This, in turn, activates IOSP and the cooperative.

When processing an input file the input cooperative determines whether the file is still input-symbiont active, and if it is, the cooperative processing waits for the symbiont processing to catch up with it on a disc-block-by-disc-block basis.

When processing an output file, the output cooperative places it in the file directory. When processing a file, the output symbiont will determine whether the file is output-cooperative active. If it is, the symbiont processing, if necessary, will wait for the cooperative processing to catch up with it on a disc-bloc-by-disc-block basis.

APPENDIX H. ERROR AND ABNORMAL RETURNS

The "error" and "abnormal" addresses specified in an FPT for a Read, Check, or Write function are temporary features and are not retained by the Monitor between calls. (Those addresses specified in an FPT for an Open function are retained in the specified DCB.)

I/O error and abnormal conditions fall into two general categories:

1. Those associated with insufficient or conflicting information.

2. Those associated with device failures or end-of-data conditions.

The Monitor responds to conditions of the first category by honoring the error and abnormal addresses in the associated DCB. The Monitor responds to conditions of the second category by honoring the error and abnormal addresses in the FPT for the associated Read, Check, or Write functions. Thus, a Read or Write function for which an error or abnormal address is specified will have an implied "wait" (for I/O completion) even if the WAIT option in the FPT is not specified.

If the user's program is to allow I/O overlap, error or abnormal returns must not be specified on I/O calls. Instead, the user's program must check (see "M:CHECK" in Chapter 5) to determine whether I/O has been completed prior to accepting data (Read functions) or using a buffer (Write functions). The error and abnormal codes for conditions of the first category, above are 01, 02, 03, 08, 0A, 13, 14, 15, 16, 17, 18, 2E, 40, 42, 43, 44, 46, 47, 4A, 51, 54, 55, and 56. Those for conditions of the second category above, are 04, 05, 06, 07, 1D, 1C, 41, 45, 49, and 57.

The Monitor communicates the error or abnormal code and the DCB address in SR3, and the location following the associated CAL1 is communicated in SR1. The code is contained in byte 0 of the word in SR3, and the DCB address is contained in bytes 1-3. The previous contents of SR1 and SR3 are lost.

The meaning of each error and abnormal code is shown in Tables H-1 through H-4.

Table H-1. Abnormal Codes – Insufficient or Conflicting Information

Hex. Code	Originating Monitor Routine	Meaning of Code Returned for Abnormal Conditions	Monitor Action if no Abnormal Address is Specified in DCB
01	OPEN	An attempt was made to open a DCB with insufficient information.	Continue job
02	OPEN	The end of all files has been encountered, and NXTF is specified in the DCB.	Continue job
03	OPEN	The "input" or "update" file does not exist.	Continue job
08	OPEN	The name of the next file was recognized as a synonym for the primary name of the file, and NXTF is specified in the DCB.	Continue job
0A	CLOSE	An attempt was made to close a DCB that is already closed.	Continue job
13	DELREC or WRITE	The specified key was not found for an "update" file.	Continue job
14	OPEN	The Monitor has not received all information needed to access the file.	Continue job
15	DELREC or WRITE	An illegal sequence of operations has been requested for an "update" file.	Continue job
16	WRITE	The NEWKEY option was specified, but the key already exists.	Continue job
17	WRITE	The NEWKEY option was not specified, for an "output" or "scratch" file.	Continue job
18	WRITE	The specified key did not conform to the file's collating sequence.	Continue job
2E	OPEN	An attempt was made to open a DCB that is already open.	Continue job

Table H-2. Abnormal Codes – Device Failure or End of Data

Hex. Code	Originating Monitor Routine	Meaning of Code Returned for Abnormal Conditions	Monitor Action if no Abnormal Address is Specified in CHECK or I/O call
04	PRECORD or READ	The beginning-of-file has been encountered.	Continue job
05	PRECORD or READ	The end-of-data has been encountered.	Continue job
06	READ	The end-of-file has been encountered.	Continue job
07	READ	Data has been lost (because the buffer was smaller than the record read).	Continue job
1C	READ, WRITE or PRECORD	The end-of-tape has been encountered.	Continue job
1D	READ or PRECORD	The beginning-of-tape has been encountered.	Continue job

Table H-3. Error Codes – Insufficient or Conflicting Information

Hex. Code	Originating Monitor Routine	Meaning of Code Returned for Error Conditions	Monitor Action if no Error Address is Specified in DCB
40	READ	A request was made to read an "output" file.	Skip to next job
42	READ or WRITE	The specified key is not valid.	Skip to next job
43	READ	No record having the specified key was found.	Skip to next job
44	WRITE	A request was made to write in an "input" file.	Skip to next job
46	READ	The DCB contains insufficient information to open a closed DCB on a Read operation.	Skip to next job
47	WRITE	The DCB contains insufficient information to open a closed DCB on a Write operation.	Skip to next job
4A	READ	The specified buffer address is not valid.	Skip to next job
51	CLOSE	A request was made to close an "output" file, prior to closing the corresponding "input" file.	Skip to next job
54	READ	The user has tried to read a control command via the C device more than once through the same DCB.	Skip to next job[t]
55	OPEN	Too many files are open simultaneously (the Monitor's file-use tables cannot handle that many files).	Skip to next job
56	CLOSE or CVOL	The disc is saturated or unable to switch to the next volume.	Skip to next job

[t] Monitor action whether an error address has been specified in the DCB or not.

Table H-4. Error Codes – Device Failure or End of Data

Hex. Code	Originating Monitor Routine	Meaning of Code Returned for Error Conditions	Monitor Action if no Error Address is Specified in CHECK or I/O call
41	READ	An irrecoverable read error has occurred.	Skip to next job
45	WRITE	An irrecoverable write error has occurred.	Skip to next job
49	WRITE	No tape unit is available.	Skip to next job
57	READ or WRITE	The disc is saturated or unable to switch to the next volume.	Skip to next job

APPENDIX I. MEMORY PROTECTION

The Monitor provides for memory protection through use of the (optional) hardware write locks and keys.[†] Table I-1 describes the lock and key settings for the different types of programs in core memory.

When the Monitor is entered, the write key control register is set to the Monitor's key code (i.e., to 00). When a foreground or background program is entered, the write key control register is set to 10 or 01, respectively.

With the above settings of the locks and keys, the Monitor has access to all of core memory. A background program may write only the background area, thereby giving the Monitor and foreground programs complete protection from background programs. Except for its data area, the Monitor is protected from foreground programs.

Table I-1. Lock and Key Settings for Programs in Core Memory

Program Type	Write Key	Locks	
		Instruction	Data
Monitor	00	11	11
Background	01	11	01
Foreground	10	11	10

[†] See "Memory Address Control" in Section 2 of the "CII 10 070 computer description manual " (C 900 950).

APPENDIX J. INFORMATION GIVEN TO USERS AND PROCESSORS

On transferring control to a user's program or to a processor, the Monitor communicates the following information via the general registers.

General Register	Information Communicated
0	TCB address

In addition to the above, processors are given the following information.

General Register	Information Communicated
2	Address of the first word location of the control command buffer.
6	Byte position (within the control command buffer) of the first byte following the name of the processor.

Example:

The address of the control command buffer is CCBUF, and the buffer contains the following data.

CCBUF	!	ƀ	S	Y	word 0
	M	B	Ȯ	L	word 1
	ƀ	L	O	,	word 2
	B	O	ƀ	ƀ	word 3
	ƀ	ƀ	ƀ	ƀ	word 4
	⋮			⋮	
	ƀ	ƀ	ƀ	ƀ	word n

In this example, register 2 contains the address of location CCBUF and register 6 contains the number 8.

APPENDIX K. CLOSING AND SAVING TAPES

CLOSED FILES

INPUT TAPES

Input tapes closed by the Monitor CLOSE routine (see M:CLOSE, Chapter 5) are always saved. The REW (rewind) option, if specified, is honored on both labeled and unlabeled tapes; the PTL (position to label) option is honored on labeled tapes only.

OUTPUT, UPDATE, AND SCRATCH TAPES

Output, update, and scratch tapes closed by the Monitor CLOSE routine are handled as indicated in Table K-1.

CLOSED VOLUMES

UNLABELED TAPES

Volumes closed on unlabeled tapes (see M:CVOL, Chapter 5) cause the tape to be rewound. The user's program has the responsibility of outputting any SAVE and DISMOUNT messages.

LABELED TAPES

Volumes closed on labeled tapes cause the tape to be rewound and a DISMOUNT message to be output. For output, update, and scratch files, a SAVE message is also output.

REWOUND FILES

UNLABELED TAPES

If the DCB associated with an unlabeled tape file is open, the tape can be rewound (see M:REW, Chapter 5). If the DCB is closed when the rewind function is attempted, the Monitor OPEN routine is called; if the OPEN is successful, the tape is rewound; otherwise, the tape is not rewound.

LABELED TAPES

If the DCB associated with a labeled tape file is open, the file is positioned to its beginning by the Monitor REW routine; otherwise, the tape is not rewound.

Table K-1. Tape Positioning for Output, Update, and Scratch Tapes

SAVE option is specified	Unlabeled tapes	OUTSN is not specified	The tape is rewound (and SAVE and DISMOUNT messages are output).
		OUTSN is specified	If REW is specified, the tape is rewound; otherwise, no action is taken.
	Labeled tapes	REW is specified	The tape is rewound.
		PTL is specified	The file is positioned to the label at the beginning-of-file; if the label is on another tape, SAVE and DISMOUNT messages are output.
		No options	No action is taken.
SAVE option is not specified	Unlabeled tapes	OUTSN is not specified	The tape is rewound and remains a scratch tape.
		OUTSN is specified	The tape is rewound and remains a scratch tape.
		OUTSN is specified	If REW is specified, the tape is rewound; otherwise, no action is taken.
	Labeled tapes	OUTSN is not specified	The tape is rewound and remains a scratch tape. Any other scratch tapes saved (due to CVOL) for the file are released for other use.
		OUTSN is specified	The tape is rewound.

APPENDIX L. LOAD MAP FORMAT

If the listing of a load map is specified in the LOAD control command for a user's program, such a map is output on the LL device when the load module is formed by the relocating loader. The format of a load map is described below.

COLUMN 1

The first column of the listing contains a code indicating the type of item referenced by the listing. The codes that may be listed are given in Table L-1.

COLUMN 2

The second column of the listing indicates the numerical value of the referenced item, in hexadecimal.

COLUMN 3

The third column indicates the byte displacement value for the referenced item.

COLUMN 4

The symbolic name of the referenced item is listed in column 4. The access code segments are also listed in this column.

If there is a Task Control Block associated with the job, it begins at the location listed (in the load map) for access code segment 0 (see Appendix O for Task Control Block format).

Table L-1. Load Map Codes

Code	Type of Item Referenced
DEF	External definition.
LDEF	Definition satisfied from library.
UDEF	Unused definition.
DDEF	Doubly defined definition.
PREF	Unsatisfied primary reference.
SREF	Unsatisfied secondary reference.
CSEC	Control section.
DSEC	Dummy section.

APPENDIX M. LOADER ERROR MESSAGES

All object module loader error messages are output on the LL device, followed by a name and a hexadecimal number. The name is that of the element file currently (or most recently) opened and the number is the sequence number of the card currently (or most recently) processed. Table M-1 lists and explains all loader error messages.

Table M-1. Loader Error Messages

UNEXPECTED EOF	An end-of-file was encountered before the end of an object module was reached (incomplete object module).
ILLEGAL RECORD I.D.	The type of record read was neither X'3C' nor X'1C' (object module) nor X'81' (library load module).
SEQUENCE ERROR	The cards of an object module were out of sequence.
ILLEGAL RECORD SIZE	The number of bytes in an object module card was less than 4 or greater than X'6C'.
CHECKSUM ERROR	A bit (or bits) was dropped in punching or reading the object module.
ABNORMAL I/O	An abnormal return was encountered while reading a library load module.
CANNOT OPEN E.F.	An element file could not be opened. (It does not exist, it has a password, etc.).
STACK OVERFLOW	Insufficient memory in which to load. If no map has been partially printed, the module is too large. If a map has been partially printed, some unsatisfied primary references have caused the stacks to grow to excessive size.
BIAS TOO LARGE	At the given bias, the load module will exceed 131K of memory.
ILL. ROM LANGUAGE	The object language in a relocatable object module was not translatable (assembler or compiler error).
BAD START ADDRESS	A start address was given which is either not on a word boundary or is not within the load module.
UNEXPECTED ROM END	Module end was given on some card of the object module other than the last card (assembler or compiler error).
0 REPEAT LOAD	An assembler or compiler generated a repeat load item with a 0 count (assembler or compiler error).
IMPROPER BOUND	A short- or long-form relocatable item was not on a word boundary.
ILLEGAL ORG	An origin was generated having no resolution or was not within the load module (assembler or compiler error).
ILLEGAL LMN	The load module file could not be opened.
SEV. LEV. EXCEEDED	The severity level specified in the LOAD card was less than that encountered in some object module or that generated by the loader (a DDEF yields a severity level of 4, a PREF yields 7).
ILL. LIB. LOAD MOD.	(PERM,LIB) was specified and the load module had one of the following: 1. More than one protection type. 2. No relocation dictionary (ABS was specified or forced by the loader due to nonstandard relocatable fields).
NO TYPE 3 PROTECTION	CSECT3 or DSECT3 was generated and the loader does not permit type 3 protection (no read, execute, or write).
ILL. DSECT	Two dummy sections having the same name but different protection types were encountered.

APPENDIX N. SPECIFIC CONFIGURING INFORMATION

PATCHING THE RESIDENT SYSTEM

The segment numbers to be used for patching the resident Monitor are as follows:

Number	Segment
0	OPEN
1	CLS
2	RDF
3	TYPR
4	IOD
5	DEBUG
6	EXIT
7	LBLT
8	OPNL
9	POS
A	CALPROC
B	WRTF
C	WRTD
D	SEGLOAD
E	KEYIN
F	LDPRG
10	PRGMLDR
11	MEMALOC
12	ALTCP
13	M:14
14	M:15
15	M:16
16	M:17
17	M:18
18	M:19
19	M:1A
1A	M:1B
1B	M:1C
1C	M:1D
1D	M:1E

FILE NAMES AND CONTENTS

File names and deck catalog numbers are given below for each file on the master tape. The listed decks should be in the indicated order within each file.

MONITOR FILES

File Name	Contents (Catalog Numbers)		
SDEBUG	704749		
ROOT	704748		
RTROOT	704969		
IOSYM	704885,	704879,	704886
IO	704750,	704751,	704752
COOP	704928, 704938	704930,	704932,
IORT	704715,	704708,	704709
HANDLERS	704368, 704371, 704766,	704369, 704373, 704753,	704370, 704773, 704754
CRDOUT	704372		
PTAP	704374		
7TAP	704851		
FBCD	704852		
DFBCD	704854		
TOPRT	704755		
PRGMLDR	704776		
TYPR	704714		
IOD	704716		
DEBUG	704756		
DUMP	704757		
EXIT	704745,	704971	
KEYIN	704746, 704963,	704929, 704020	704762,
M:14	704972		
M:15	704931, 704937	704936,	704964,
M:16	704934		
M:17	704935		
M:18	704973,	704894,	704968
M:1A	704761,	704895	

File Name	Contents (Catalog Numbers) (cont.)
M:1B	Dummy file
M:1C	Dummy file
M:1D	Dummy file
RDF	704713, 704718
OPNL	704712
M:1E	704866
OPN	704710
CLS	704722, 704882, 704892, 704878, 704887, 704888, 704889, 704890
SEGLOAD	704763, 704764, 704880
LDPRG	704744
MEMALOC	704759 704760, 704767
CALPROC	704758
WRTF	704721
WRTD	704719
DUMMYCCL	704883
CCLOSE	704933
LBLT	704717
M:19	704723
POS	704720, 704881, 704966
ALTCP	704765
OBSE	704711
CLS1	704891, 704884
RTROOT	704969

LOADER FILES

File Name	Contents (Catalog Numbers)
LDR	704724
IN1	704725
PS1	704726
IN2	704727
PS2	704728
ALL	704729
EVL	704730
WRT	704731
MODIFY	704898

CCI FILES

File Name	Contents (Catalog Numbers)
CCIROOT	704901, 704732, 704902, 704903, 704735, 704904, 704733
M:DLIMIT	704736
JOB	704905
LIMIT	704906
ASSIGN	704734, 704907
LOAD	704738
TREE	704741
TELSCPE	704742
RUN	704743
CCIDBUG	704737
READBI	704908
ENDJOB	704909
ABORT	704910

TAPEFCN FILES

File Name	Contents (Catalog Numbers)
TAPEFCN	704739
TPECHST	704740

FMGE FILES

File Name	Contents (Catalog Numbers)
FILEMNGE	704747
FMGEDCBS	704874
CHKPTROM	704023

LIBRARY FILES

File Name	Contents (Catalog Numbers)
M:CDCB	704911
M:OCDCB	704912
M:LODCB	704913
M:LLDCB	704914
M:DODCB	704915
M:PODCB	704916
M:BODCB	704917
M:LIDCB	704918
M:SIDCB	704919
M:BIDCB	704920
M:SLDCB	704921
M:SODCB	704922
M:CIDCB	704923
M:CODCB	704924
M:ALDCB	704925
M:EIDCB	704926
M:EODCB	704927
M:GODCB	704941
M:CKDCB	704058
BDBROM	704031
SSSROM	704032

SYSTEM GENERATION PROCESSORS

File Name	Contents (Catalog Numbers)		
PASS1ROM	704867,	704877,	704939
DEFROM	704876,	704875,	704873
PASS2	(See segments listed below)		
CCLOAD	704899		
PASS2CCI	704896		
DCBS	704940		
SGLDR	704959		
DEVICE	704897		
SDEVICE	704893		
MONITOR	704868		
DLIMIT	704957		
LIBLOAD	704900		
DLIB	704870		
FORMLIB	704869		
INTSR	704958		
RESERVE	704871		
RJITGEN	704872		
CLOCK	704967		

TREE STRUCTURES

MONITOR

The tree structure of the Monitor is given in Figure N-1. The options that should be specified in the OVERLAY command used to form the Monitor are shown in the following example.

```
!OVERLAY  (BIAS,0),(NOTCB),(ABS),(SL,F),;
!(NOSYSLIB),(PERM),(LMN,M:MON)[,(MAP)]
```

As indicated in Figure N-1, the tree structure may be varied slightly to suit the requirements of the installation. If a non-symbiont system is desired, the IO and DUMMYCCL files should be included; otherwise, the IOSYM and CCLOSE files (as well as M:SDEV and COOP) should be used instead. If the Monitor is to handle any foreground tasks other than directly entered real-time programs, then RTROOT, M:RJIT, M:NRJIT, M:RESDF, M:INT, and M:TIME must be included. If foreground COMMON is desired, M:FCOM must be included also. M:RESDF must be included if patches are to be made to the Monitor. Handlers for standard system I/O devices (i.e., KBTIO, CRDIN, PRTOUT, MTAP, and DISCIO) are included in the HANDLERS file; three others (i.e., CRDOUT, PTAP, and 7TAP) are optional and must be specified individually in the TREE command, if desired. If FORTRAN BCD conversion capability is desired, FBCD must be included; otherwise, DFBCD should be selected. Any appropriate user initialization routines may be included, as desired (i.e., USRINIT1, USRINIT2, USNRINIT1, and USNRINIT2).

LOADER

The tree structure of the loader is given in Figure N-2. The options that should be specified in the OVERLAY command used to form the loader are shown in the following example.

```
!(LMN, LOADER)[, (MAP)]
!OVERLAY  (BIAS, value), (NOTCB), (PERM),;
```

The bias value specified should range from 1400 (minimum system) to 2400 (maximum system).

CONTROL COMMAND INTERPRETER

The tree structure of the control command interpreter is given in Figure N-3. The options that should be specified in the OVERLAY command used to form the control command interpreter are shown in the following example.

```
!(LMN, CCI), (PERM)[, (MAP)]
!OVERLAY  (BIAS, 2400),  (NOTCB),;
```

PASS2 PROCESSOR

The tree structure of the PASS2 processor is given in Figure N-4. The options that should be specified in the OVERLAY command used to form the processor are shown in the following example.

```
!(LMN, PASS2), (TSS, 800)[, (MAP)]
!OVERLAY  (BIAS, 2400), (PERM), (SL, F),;
```

UN-SEGMENTED PROCESSORS AND DCB'S

The following are all simple program structures and are formed by the use of LOAD commands rather than OVERLAY commands.

PASS1 PROCESSOR

The LOAD command used to form the PASS1 processor is shown below.

```
!(TSS, 100), (BIAS, 2400)[, (MAP)], (PERM)
!LOAD  (LMN, PASS1), (EF, (PASS1ROM)),;
```

DEF PROCESSOR

The LOAD command used to form the DEF processor is shown in the following example.

```
!(TSS, 100), (BIAS, 2400)[, (MAP)], (PERM)
!LOAD  (LMN, DEF), (EF, (DEFROM), (M:LLDCB)),;
```

FMGE PROCESSOR

The LOAD command used to form the FMGE processor is shown below.

```
!(FMGEDCBS)), (TSS, 100), (BIAS, 2400)[, (MAP)]
!LOAD  (LMN, FMGE), (EF, (FILEMNGE), (TPECHST),;
```

REW, WEOF, AND PFIL PROCESSORS

The LOAD command used to form any one of these processors is shown below.

```
![, (MAP)], (PERM)
!(M:GODCB),(TSS,100),(BIAS,2400),;
!(M:ALDCB),(M:EIDCB),(M:EODCB),;
!(M:SODCB),(M:CIDCB),(M:CODCB),;
!(M:SIDCB),(M:BIDCB),(M:SLDCB),;
!(M:PODCB),(M:BODCB),(M:LIDCB),;
!(M:LODCB),(M:LLDCB),(M:DODCB),;
!(TPECHST),(M:CDCB),(M:OCDCB),;
!LOAD (LMN,name),(EF,(TAPEFCN),;
```

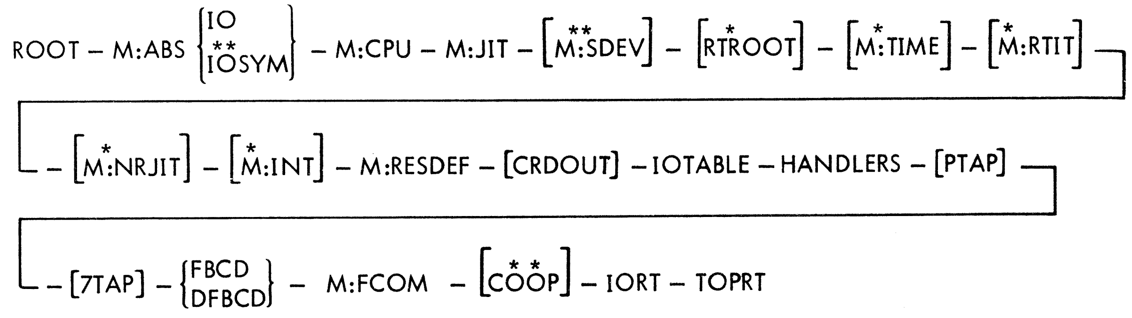
The "name" identifying the load module in the above example is REW, WEOF, or PFIL, as appropriate.

SYSTEM DCB'S

The LOAD command used to form any one of the system DCBs is shown below.

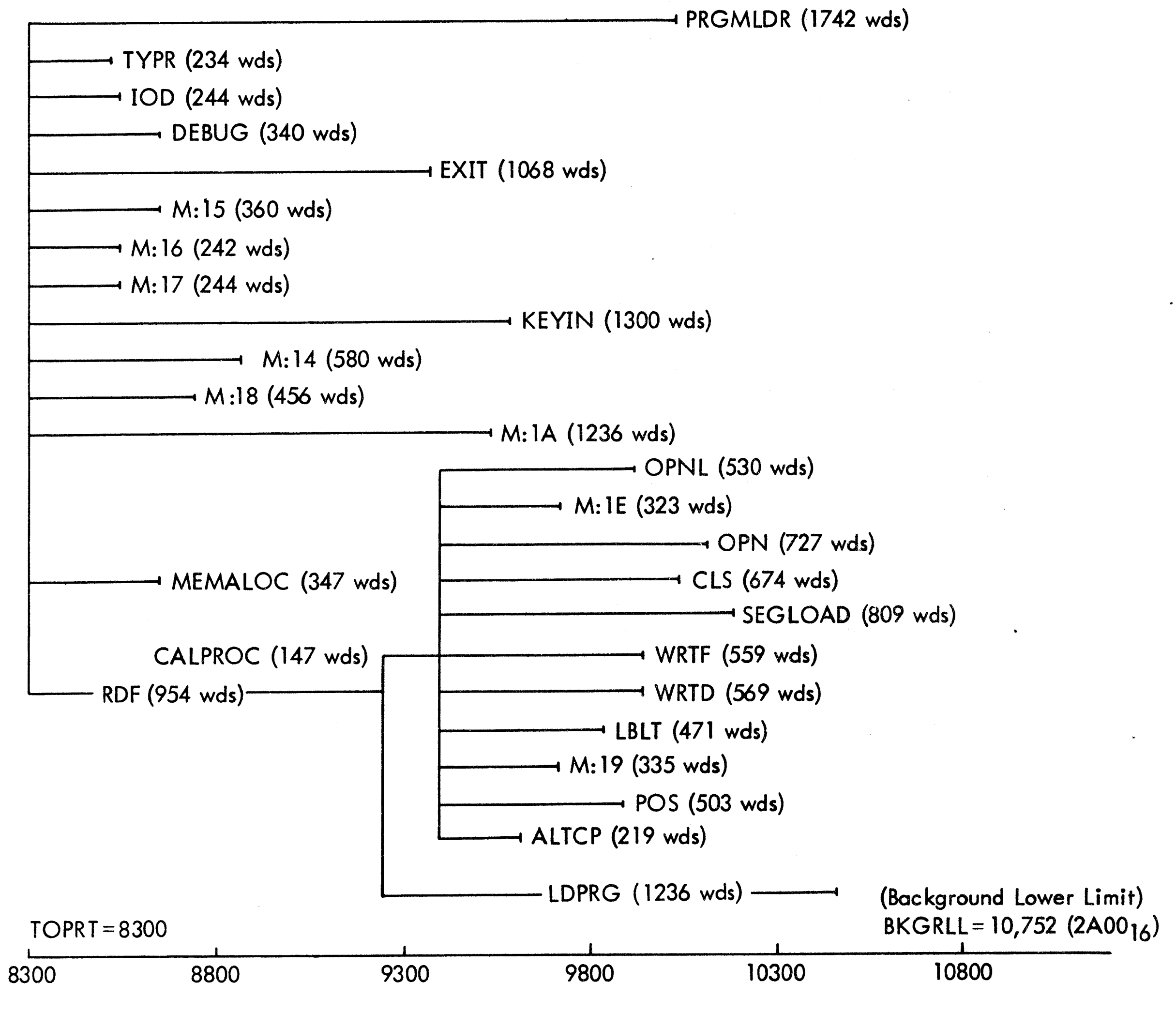
```
!(PERM, LIB)[, (MAP)]
!LOAD  (LMN, name), (EF(file)),;
```

The "name" identifying the load module in the above example is the name of the system DCB (e.g., M:C); "file" is the name of the corresponding file in the system library (e.g., M:CDCB).


ROOT – M:ABS {IO / **IOSYM} – M:CPU – M:JIT – [**M:SDEV] – [*RTROOT] – [*M:TIME] – [*M:RTIT]
– [*M:NRJIT] – [*M:INT] – M:RESDEF – [CRDOUT] – IOTABLE – HANDLERS – [PTAP]
– [7TAP] – {FBCD / DFBCD} – M:FCOM – [**COOP] – IORT – TOPRT

*Real-time system options.
**Symbiont system options.

Figure N-1a. Monitor Root Structure



Note: This figure shows the BPM overlay tree (with sizes in decimal words) of a symbiont real-time system including:

8 symbiont buffers (256 wds)	1K wd temp stack
6 symbiont context buffers (40 wds)	3 discs
10 Monitor print buffers	power fail-safe option
12 queue entry blocks	100 wds patch space

Figure N-1b. Monitor Tree Structure

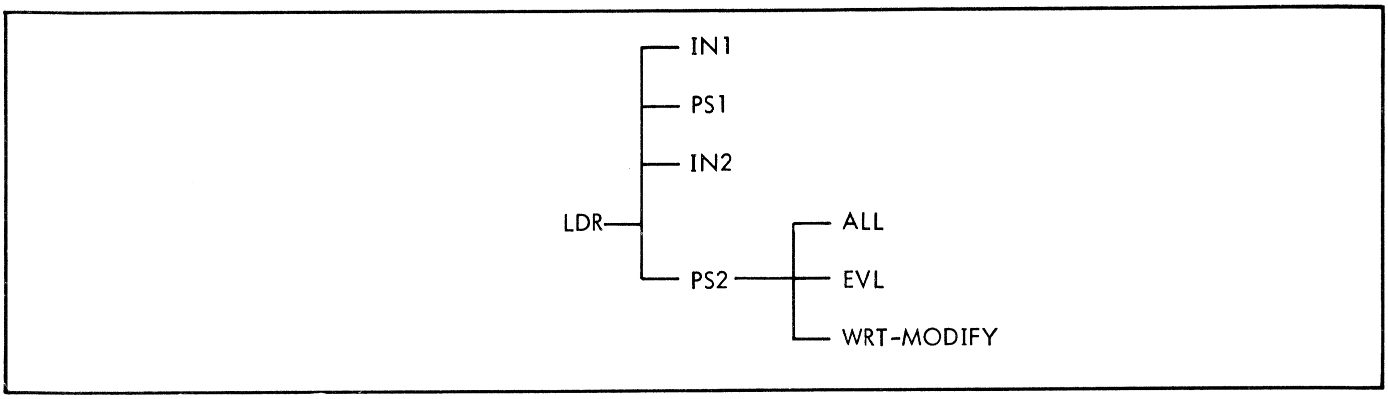


Figure N-2. Loader Tree Structure

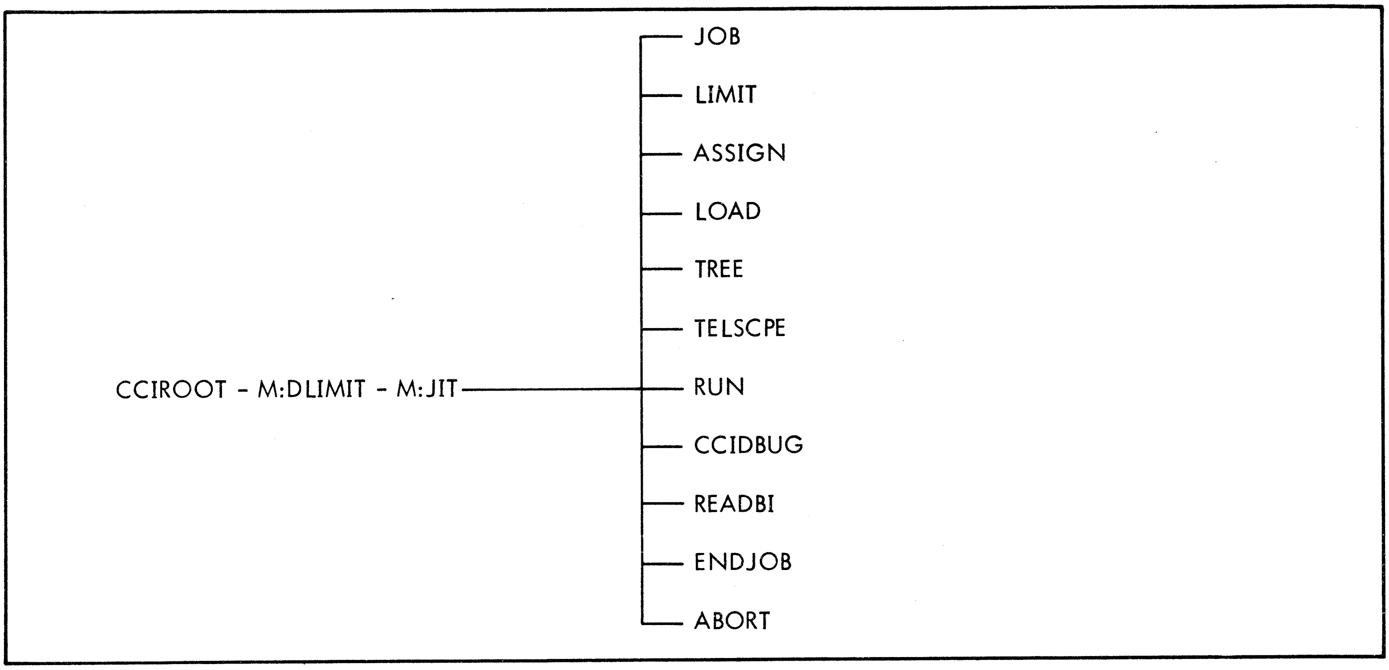


Figure N-3. Control Command Interpreter Tree Structure

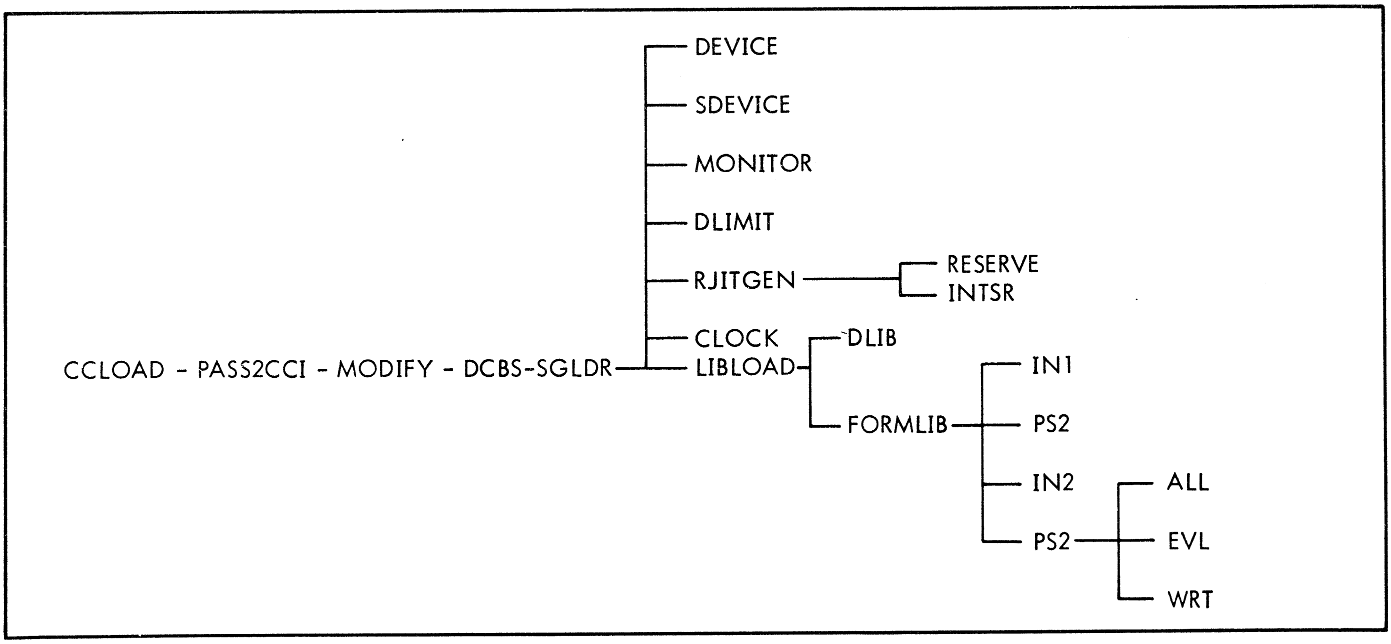


Figure N-4. PASS2 Processor Tree Structure

APPENDIX O. TASK CONTROL BLOCK FORMAT

The format of the TCB established and maintained by the Monitor for the user's program is shown below.

0	0————0	TSTACK	
1	TSS	0————0	
2			
3	These words for use by processor		
4			
5			
6	0————0	TSA	
7	TSASIZ	0————0	
8	ERTSIZ	ERT	
9	ERTSIZ-2	TSA+1	
10	0————0	DCBTAB	
11	0————0	TREE	
12	0————0	SSW	
13	For use by Processor		
14	XSL specified on !RUN card		
15	For use by Monitor		
TSA	Library error temp stack		} TSASIZ
ERT	Library error table		} ERTSIZ
TSTACK	User's temp stack		} TSS
	0 14 15 16	25 26 31	

where

TSTACK is the address of the current top of the user's temp stack.

TSS indicates the size, in words, of the user's temp stack.

TSA is the address of the temp stack used by the library error package.

TSASIZ indicates the size, in words, of the temp stack used by the library error package.

ERTSIZ indicates the size, in words, of the error table used by the library error package.

ERT is the address of the error table used by the library error package.

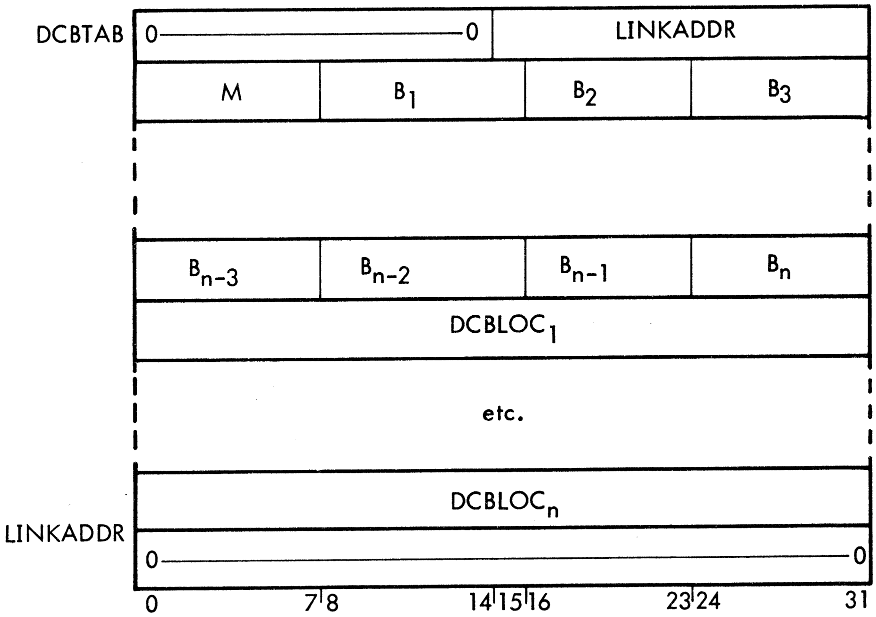
DCBTAB is the address of a table of names and addresses of all of the user's DCBs. This table has the form shown below under "Table of User's DCBs".

TREE is a pointer to the location of the user's overlay structure.

SSW contains the user's sense switch settings (bit 26 contains the setting of switch 1, etc.).

TABLE OF USER'S DCBs

The table of user's DCBs has the following format.



where

LINKADDR is the address of the location provided for storing a return address.

M indicates the number of characters in the DCB name.

B_1-B_n indicates the EBCDIC name of the DCB.

DCBLOC is the address of the first word location of the DCB.

APPENDIX P. FILE ATTRIBUTES

A discussion of several of the attributes of files under the BPM will help the user to more profitably make use of this part of the BPM.

It is important to understand the inter-workings of DCBs, !ASSIGNs and M:OPENs. File attributes may be specified on any or all of these items. Attributes specified on !ASSIGN override those built into DCBs. In turn, attributes expressed on M:OPEN override those specified by either DCBs or !ASSIGNs.

In some cases, attributes control the file itself rather than its usage. In these cases the correct attributes are placed into the DCB when it is opened. For example, if the user creates a file ALPHA as a KEYED file and tomorrow reads the file specifying CONSECutive organization, BPM will override the CONSECutive parameter with a KEYED parameter. Thus, for example, it is never meaningful to specify organization when opening an INput or INOUT (update) file.

Four parameters specified by the user in DCBs on !ASSIGN cards, and/or by OPEN or CLOSE commands control file use:

1. Function – relates to information direction IN, OUT, etc.
2. Disposition – controls the saving and releasing of files both within the job and at its termination.
3. Organization – tells how the file is put together.
4. Access – tells how the file should be accessed.

FUNCTION

This attribute specifies whether the file is to be read or written, and (by implication) whether an existing file is to be accessed or a new file is to be created. Function also controls the disposition of the file (see "Disposition Type").

INput specifies that an existing file is to be accessed for reading only.

OUTput specifies that a new file is to be created.

INOUT specifies that an existing file is to be accessed for reading and writing (updating).

OUTIN specifies that a new file is to be created but it may also be read before closing.

A file opened as OUT or OUTIN does not replace a previous file by the same name until it is saved when the CLOSE command is given. Thus two files of that name exist during creation.

Several DCBs may be simultaneously open to the same file (and/or the same file name) if the proper protocol is observed. Note that a DCB open to file ALPHA as IN and another DCB open to file ALPHA as OUT are not referencing the same file (merely same file name). For DCBs open to the same file name the following sequences of opens are permissible:

1. DCB1,IN ... DCB2,IN ... DCB3,IN ...
2. DCB1,OUT ... DCB2,IN ... DCB3,IN ...
3. DCB1,OUTIN ... DCB2,IN ... DCB3,IN ...

In cases 2 and 3, DCB2 ... DCBN must be closed before DCB1 may be closed and saved. Otherwise the new output file is released.

Thus to generalize, if a DCB is opened INOUT to a file, no other DCB may be opened to the same file name. If a DCB is opened IN, OUT, or OUTIN, other DCBs may be opened IN only. A DCB opened as OUT or OUTIN may be closed and saved only if there are no other DCBs open to the same file.

If the above protocol is violated, an abnormal code of 14 is returned for an Open command.

DISPOSITION TYPE

There are two levels of disposition types available in BPM. The user may first specify whether the file is to be SAVEd or RELeased when the controlling DCB is closed. If a file is SAVEd, it may be declared to be either TEMPorary or PERMAnent, causing it to be released or retained at the end of the job. Thus REL/SAVE are concerned only with the current use (open DCB) of the file while PERM/TEMP are job associated. A temporary file may be opened and closed many times before it disappears at the end of the job.

All IN or INOUT files are saved unless explicitly RELeased on an M:CLOSE operation. All OUT and OUTIN files are released unless SAVE is specified by the merged parameters from DCB, ASSIGN, M:OPEN, and on M:CLOSE. If any DCBs remain open when a program exits (or errors or aborts), they are closed with neither SAVE nor REL specified. Thus, currently open IN (or INOUT) files are saved and currently open OUT (or OUTIN) files are released. The following job will normally replace the old copy of MYCI with the new MYCI when the assembly is complete. If the assembly is terminated prematurely for any reason, the partially created MYCI is released and the old MYCI is retained.

```
!JOB         MYACCT,MYNAME
!ASSIGN      M:CI,(FILE,MYCI)[†]
!ASSIGN      M:CO,(FILE,MYCI)[††]
!METASYM     SI,CI,CO,LO,GO
updates
+END
```

[†] CI DCB has IN built in.

[††] CO DCB has OUT and SAVE built in.

The TEMPorary files to be released at the end of the job are remembered in a list of names declared temporary. Once a file name is declared temporary, it cannot be removed from the list. Thus declaring a file temporary, then later declaring it permanent results in a temporary file.

Files may be declared TEMPorary in several ways:

1. M:TFILE
2. absence of PERM on :FMGE (ENTER)
3. absence of PERM on !LOAD
4. any file created as the result of GO specified on a processor card even if M:GO is assigned. This allows users to use the GO option to create an overlay structure, while keeping the element files as temporary files.

ORGANIZATION

CONSECutive organization implies that the records within a file are order dependent, that the file was created SEQUENTially and that the records may only be accessed SEQUENTially.

KEYED organization implies that each record in a file has a unique identifier (key) which was supplied when the file was created. If the records were presented in random order, they have been sorted alphanumerically (by Key) so that subsequent SEQUENTial access will retrieve the records in sorted order. The file may also be accessed DIRECTly, accessing each record by name (Key).

ACCESS

SEQUENTial access implies that the records are to be accessed in order. Positioning operations are valid. This access type may be used for either CONSEC or KEYED files.

DIRECT access implies that all records are to be accessed by name (Key). This may be used with KEYED files only.

APPENDIX Q. CHARACTER ORIENTED COMMUNICATIONS ROUTINES

The Character Oriented Communications (COC) routines are resident real-time programs for communicating with 70 015 printers, and CII keyboard displays via a single 70 611 communication controller A user's program loaded with the COC routines receives control when a complete message is received from a terminal. Control is returned to the batch background operation when a Read command is issued to the COC routines but no complete message is ready. Control returns to the COC routines and thus to the issuing program when a complete message is available.

Control of the console is proprietary: i.e., input is not allowed during output and output is not allowed during input. Communication is, however, available in both directions, and the receipt of the "break" character is signaled to a program doing write operations.

Characters going to and from the terminals are translated to and from internal form through tables associated with individual lines. These tables also control end-of-message action and special character functions. Special tables tailored to other consoles or to other end-of-message characters can be supplied to the COC routines through reassembly, or dynamically through direct access to line control information.

COMMANDS

Four commands interface the user's program with the COC routines: ACCEPT INPUT, READ, WRITE, and WAIT FOR OUTPUT. The ACCEPT INPUT command directed to a specified terminal readies that terminal for input, supplying a buffer in which received characters are to be placed. READ returns a complete message with its length in bytes and the number of the terminal from which it came. If no complete message is available, control returns to the current batch job. The WRITE command stacks a message for output to a specified line. The WAIT FOR OUTPUT command gives up control to batch operation until some line completes output. Condition codes report invalid line numbers, output attempted to a line currently accepting input, and receipt of a "break" character from the terminal since the last call on the COC routines.

LINE STATES

There are five line states:

INAC Inactive line.

OUT Output. The computer is transmitting characters to the line.

IN Input. The computer is receiving characters from the line.

IC Input complete. An end-of-message[†] or activation character has been received from the line. The completed message will be delivered to the user program when a READ command is given.

SI Switch to input. The computer is transmitting to the line; input will be accepted as soon as output is complete.

Input characters are ignored unless the line state is IN. Output occurs only in OUT or SI states.

HANDLING OF THE BREAK SIGNAL

The special line signal "break" is recognized and recorded regardless of line state or condition of transmission. When a break signal is received, a bit in the line control table associated with the sending terminal is set. The condition of this bit is reported to the user on a Read or Write operation and the bit is cleared. During input from the terminal, the receipt of a break signal is also reported by placing the character EOT in the input buffer and setting the end-of-message or "activate" condition. If the program using the COC routines wishes to examine the break condition without doing a Read or Write, it may examine the bit directly (see "Line Control Tables".).

INPUT PROCESSING – TTY

The standard Teletype conversion tables (Tables Q-1 and Q-2) provide for input line editing, COC mode control, and message-complete characters as follows.

Cr	When the carriage return (Cr) is received, a line feed (Lf) is sent to the terminal, the message is marked complete, and the associated program is activated if it was waiting at a READ command. Further input from the terminal is ignored until an ACCEPT INPUT command is received.
Lf	Same as Cr except that Cr is sent to the terminal.
Cancel (CAN, X^c)	When the character CAN (generated at the Teletype by pressing the control and X keys) is received, the current partial input message is erased. The characters ← Cr Lf are sent to the terminal.
Rubout	When the character "Rubout" is received the preceding character is erased and a blank is sent to the terminal. Cr and Lf are sent if the entire input message is erased in this way.
US	When the US character (generated by pressing both shift keys and the O key) is received, the bit controlling character echoing is complemented. This bit, when set, causes the COC routines to

[†] In the standard Teletype case, the end-of-message characters are Cr, Lf, ESC, and break.

	echo received characters back to the terminal to cause printing at terminals whose keyboards are not locally connected to the printer.
ALTMODE or ESC	The message-complete condition is set.

INPUT PROCESSING – K/D

In general, end-of-message activation occurs on all cursor movements, the hard copy control characters, the roll characters, and the mode change characters.

RS	When the RS character (generated by pressing both shift keys and the N key) is received, the bit controlling the keyboard/display local mode is complemented. This bit, when set, causes the COC routines to transmit, or echo, received characters to the terminal but take no further action.

OUTPUT PROCESSING

Standard output conversions for Teletypes and keyboard/displays are shown in Table Q-3. The tables give the 7-bit ASCII character sent for each EBCDIC character. In each table is an eight-bit value including parity for transmission to the terminal. Illegal characters are not sent to the terminal. For Teletype terminals, both Cr and Lf characters are sent as the pair CrLf; lower case EBCDIC alphabetics are sent in upper case; and message output is terminated by a zero byte or by byte count.

CREATION AND INITIALIZATION OF A RESIDENT REAL-TIME COC SYSTEM

In order to operate a COC foreground system, the user must proceed as follows.

1. Generate a system with core and interrupt locations reserved.
2. Assemble the standard COC routines, specifying the correct number and types of lines, number of buffers, conversions desired, etc.
3. Create a load module on file including the COC routines and the user's program.
4. Cause the load module to be placed in core ready to run.
5. Trigger initial operation of the process via a console command.

SYSTEM GENERATION

The generated BPM system must provide for the real-time COC capability. This is accomplished in PASS2 of System Generation by specifying appropriate options in the RESERVE and INTR control commands for core and interrupt location assignment. Additional options and control commands may be needed as outlined in Section 10 of this manual.

:RESERVE Command Specifications for COC

Core size specified on the RESDF option must be sufficient to support the COC routines (about 2000 words) and user program. FIPOOL and FFPOOL options should appear to supply an appropriate number of buffers, if any file I/O is to be performed by the user's program.

:INTR Command Specifications for COC

This control command is used to specify the load module name of the foreground COC task and its associated interrupt location. Since there are two interrupts associated with the COC system, both must be specified. In addition, the COC system operates in the master mode and therefore the MASTER option must be specified.

COC ASSEMBLY PARAMETERS

The COC system must be assembled and a relocatable object module (ROM) generated that represents the user's needs. That is, the following parameters must be defined by the user in the COC source deck:

COCNB	Number of buffers to be provided (each buffer is 4 words long – enough for 14 characters).
COCDN	Device number of attached COC.
COCNL	Number of lines connected to COC (it is assumed that the connected lines are numbered from zero upward to COCNL-1, and 64 is the maximum for the current implementation).
COCII COCIO	Locations of input and output external interrupts.

In addition, the user must specify the locations of the input (COCIT) and output (COCOT) conversion tables associated with each line, as well as the initial line mode (MODE). The standard COC routines are assembled with I/O conversion tables for Teletypes connected for local printing.

Certain information is collected and used by the COC routines during execution. This information is available to the user's program. The externally defined and referenced symbols and their contents are described below.

Symbol	Type	Contents
COCBW	(Def)	Loop count waiting for buffers
COCIPC	(Def)	Input parity error count
COCIPL	(Def)	Input parity error line number (last line)
COCILC	(Def)	Input logical error count
COCILL	(Def)	Input logical error line number (last line)

Symbol	Type	Contents
COCUP	(Ref)	Location of user's program initialization entry
COCOEC	(Def)	Output extraneous interrupt count
COCOEL	(Def)	Output extraneous interrupt line number (last line)

Line Control Table Entry

Symbol	Type	Contents	Size
STATE	(Def)	Line state	(byte)
MODE	(Def)	Line mode	(byte)
COCOT	(Def)	Location of output conversion table	(word)
COCIT	(Def)	Location of input conversion table	(word)
LINK	(Def)	Relative buffer address	(half)
COCBA	(Def)	Byte address of next input or output character	(word)
COCOCNT	(Def)	Output character count	(half)
ARS	(Def)	Input character count	(byte)

CREATING A COC SYSTEM FILE

The COC system is formed under Monitor control by running a job under the account number :SYSRT. The Monitor control command LOAD is used to direct the loader to form a relocatable load module from the COC and user's relocatable object modules. The resulting program is placed in the real-time account file.

LOADING CORE WITH THE COC SYSTEM

The RUN control command is used to enter the load module (created by the LOAD control command) into the resident foreground area, and to connect the COC system to its associated interrupt (input). The name of the load module (LMN, name) must correspond to the name specified on the INTR control command. Any options specified in the RUN control command must agree with the System Generation specifications. The DIRECT option should be used; it specifies that the foreground task (COC system) is to be entered directly each time the associated interrupt becomes active. The MASTER and RESD options must have been specified on the INTR control command at System Generation to validate the option. The task is entered in the master mode and is responsible for saving and restoring any machine environment it changes. It is also responsible for returning control to the point at which the interrupt occurred.

INITIATION OF THE COC SYSTEM

The unsolicited TRIGGER key-in is used to trigger the external interrupt. If the interrupt is not armed, the trigger has no effect.

The format of the key-in is

```
!TRIGGER location
```

where "location" specifies the hexadecimal location of the external input interrupt. When the interrupt occurs, the COC routines initialize themselves and the assigned terminal lines. Control is then passed to the user's program for initial operation.

RESTRICTIONS

FOREGROUND FILES

A real-time foreground task may not create new files but may open a file in any mode; similarly, it may not release files. Foreground tasks are restricted to updating or reading files from their own account and reading files from other accounts. A batch job running under a foreground account may prevent a foreground job from updating a file if it has the file open when the foreground wants access. If this occurs, an abnormal code of 14 will be communicated in SR3 at the time the file is accessed.

A batch job running under a foreground account may add files to the foreground file directory, but in so doing will prevent foreground jobs from obtaining access to the directory. If this occurs, an abnormal code of 20 is communicated to the foreground task in SR3 when the file is accessed. As a general rule, a batch job should not update or create foreground files if the file directory is in danger of being used by a foreground task.

PROCEDURE CALLS

The following Monitor procedure calls will be ignored if made by resident foreground programs.

```
M:CHKPT      M:GCP
M:RESTART    M:FCP
M:LDTRC      M:GP
M:LINK       M:FP
M:GL         M:SMPRT
```

COC COMMAND DETAILS

The routines defined here follow the conventions of library routines in that they destroy the contents of registers 6-11. The following routines are provided.

COCRD (READ MESSAGE)

The linkage

```
BAL,11 COCRD
```

causes the COCRD routine to select the first complete message and move that message to the requested area. The number of bytes moved will be the minimum of the actual record size (number of characters contained in the message) and the size of the message requested. The COC assumes the following register settings:

General register 7	Byte address of the area in which the input message will be packed (left-justified).
General register 8	Maximum byte size of the message to be read (right-justified).

Upon return to the user's program, the following information is available:

General register 6	Binary line number from which the message came.
General register 8	Size, in bytes, of the message transferred.

Condition Code 1 is set if a break character was received.

COCWR (WRITE MESSAGE)

The linkage

BAL,11 COCWR

causes the COC to initiate an output message on the specified terminal if not already in the output state. The results of the operation may be tested for an invalid line number, the receipt of a break character, or an invalid request. The Write call causes the line state to be changed from inactive to output. If the line state is not inactive or output, the request will be ignored and Condition Code 2 set. Condition Code 1 is set if a break code has been received. In honoring the request, the COC routine will not return control to the user's program until enough buffer storage is obtained to move the message from the user's area to internal blocking buffers. The COC assumes the following register settings.

General register 6	Line number upon which the message is to be transmitted.
General register 7	Byte address of the message to be transmitted.
General register 8	Byte count of the message to be transmitted. The message is terminated on a zero byte in the standard TTY conversion.

Upon return to the user's program, the following condition codes apply.

Condition Code				Meaning
1	2	3	4	
0	0	0	0	Normal transfer.
0	1	0	0	Invalid line state (not inactive or output); request ignored.
1	0	0	0	Break signal received.
0	0	1	0	Invalid line number; request ignored.

COCAI (ACCEPT INPUT)

The linkage

BAL,11 COCAI

causes the COC to accept input from the specified terminal. If the terminal is in the output state, the state will be switched to input when the last character of the output message is transmitted. If the terminal state is inactive, the state will be changed to input. If the terminal state is input or an input message is already complete, the request is ignored and CC2 is set.

The COC assumes the following register setting.

General register 6	Line number on which input is requested.

Upon return to the user's program, the following information is available in the Condition Codes.

Condition Code				Meaning
1	2	3	4	
1	0	0	0	Break signal received.
0	0	1	0	Invalid line number; request ignored.
0	1	0	0	Invalid line state; request ignored.

COCREL (WAIT FOR OUTPUT)

The linkage

BAL,11 COCREL

returns control to the Monitor until output has completed on some line (a line state change from OUT to INAC). The number of the first line completing output is returned in register 6 when control returns to the instruction following the BAL. Condition Code 1 is set if a break character was received on that line. If no lines are in the OUT state when COCREL is entered, exit is immediate with Condition Code 2 set.

COCTR (TERMINATE COC I/O)

The linkage

BAL,11 COCTR

causes all COC I/O to be terminated, all interrupts disarmed, and control returned to the interrupted batch program.

LINE CONTROL TABLES

The line control tables contain information regarding lines to and from terminals. This information pertains to status, mode, and input/output conversion.

STATE TABLE – STATE (BYTE TABLE)

Byte	State	Meaning
0	INAC	inactive – not accepting or transmitting characters
1	IN	accepting input
2	OUT	transmitting output
3	IC	input message complete
4	SI	line is to be activated for input after output is complete

INPUT CONVERSION TABLE LOCATION – COCIT (WORD TABLE)

COCIT is a 128-byte table. The input characters (7 bits) received from the line index the byte table. The contents of the table entry determine the conversion of the input character.

MODE TABLE – MODE (BYTE TABLE)

Bit	Name	Value	Meaning
0	echo	0	Do not echo (terminal is local printing)
		1	Echo (echoplex terminal)
1	K/D	0	K/D not local
		1	K/D local
2	bk	0	Break not received
		1	Break received
5, 6, 7	special output flag	000	No special characters to be transmitted
		001	Cr to be transmitted
		010	Lf to be transmitted
		011	Cr and Lf to be transmitted
		100	End of special transmission

OUTPUT CONVERSION TABLE LOCATION – COCOT (WORD TABLE)

The output table is a 256-byte output conversion table. Output characters received from the user's program index this table. The contents of each entry represent the character to be transmitted, including parity (if appropriate). When the entry contains F1, F2, or F4, special action is taken (see "Output Special Character Routines").

ACTUAL RECORD SIZE TABLE – ARS (BYTE TABLE)

Actual record size is the count of all converted input characters contained in the current message.

BYTE ADDRESS FOR NEXT CHARACTER TABLE – COCBE (WORD TABLE)

The buffer pointer is the byte address at which the next input character is to be placed, or from which the next output character is to come.

LINK TABLE – LINK (HALF WORD TABLE)

The LINK table contains the relative buffer address (buffer address minus buffer pool location) of the first message buffer. If more than one buffer is used for I/O, the buffers are linked. The last buffer has a zero link. The first halfword of each buffer is used for linking, as shown in buffer format below.

OUTPUT CHARACTER COUNT TABLE – COCOCNT (HALFWORD)

The COCOCNT table contains the count of characters remaining to be transmitted to the terminal.

LOCAL STORAGE

Local storage consists of a group of variables and buffers used in the storage and manipulation of characters and character strings.

COCBUF Word address of buffer pool.

COCHPB Head of the available buffer pool. Each available buffer is linked in its first halfword to the next. The last available buffer has a zero link. The links are displacements from the beginning of the buffer pool.

COCINRES n words of temporary storage used by the input character interrupt routine.

COCOUTRES m words of temporary storage used by the output character interrupt routine.

COCRES 18 words of temporary storage used to retain the interrupted environment. The interrupted environment is saved in this area when control is transferred to the user's program.

COCPS Program state:

= 0 User's program is currently inactive.

≠ 0 User's program is currently active.

BUFFER FORMAT

word 0	Link		C_1	C_2
word 1	C_3	C_4	C_5	C_6
word 2	C_7	C_8	C_9	C_{10}
word 3	C_{11}	C_{12}	C_{13}	C_{14}

where

link contains the relative address of the next buffer (buffer address – COCBUF).

C_i is the I/O message character.

INPUT SPECIAL CHARACTER HANDLING ROUTINES

In the input conversion tables, values in the EBCDIC column from 30 to 3A require the special handling routines listed below.

30 Cr – echo Lf; activate; X'D' goes to input buffer

31 Lf – echo Cr; activate; X'15' goes to input buffer

32 RUBOUT – echo " " backup pointer; echo Cr Lf if beginning

33 2O (US) – Toggle TTY ECHO mode

34 2N (RS) – Toggle K/D ECHO mode

35 X^c (CAN) – "   Cr Lf"; release all but one buffer

36 Ignore character

37 Count error and ignore

38 Count error and ignore; put X'FF' in buffer and echo "#"

39 Echo; put ASCII character in buffer and activate

3A ESC or ALTMODE – activate; X'FE' goes to buffer

Table Q-1. Input Conversion – TTY

ASCII		EBCDIC				ASCII		EBCDIC		
Code	Character	Code	Character	EOM	Note	Code	Character	Code	Character	EOM
0	NULL	00				20	blank	40	blank	
1	SOH	01				21	!	5A	!	
2	STX	02				22	"	7F	"	
3	ETX	03				23	#	7B	#	
4	EOT	38				24	$	5B	$	
5	ENQ	09				25	%	6C	%	
6	ACK	06				26	&	50	&	
7	BEL	07				27	'	7D	'	
8	BS	18				28	(	4D	(	
9	HT	05				29	)	5D	)	
A	NL	31		X	(1)	2A	*	5C	*	
B	VT	0B				2B	+	4E	+	
C	FF	0C				2C	,	6B	,	
D	CR	30		X	(2)	2D	-	60	-	
E	SO	0E				2E	.	4B	.	
F	SI	0F				2F	/	61	/	
10	DLE	10				30	0	F0	0	
11	DC1(XON)	11				31	1	F1	1	
12	DC2(TAPE)	12				32	2	F2	2	
13	DC3(XOFF)	13				33	3	F3	3	
14	DC4($\overline{\text{TAPE}}$)	14				34	4	F4	4	
15	NAK	0A				35	5	F5	5	
16	SYN	16				36	6	F6	6	
17	ETB	17				37	7	F7	7	
18	CAN	35			(3)	38	8	F8	8	
19	EM	19				39	9	F9	9	
1A	SS	1A				3A	:	7A	:	
1B	ESCI	1B				3B	;	5E	;	
1C	FS	1C				3C	<	4C	<	
1D	GS	1D				3D	=	7E	=	
1E	RS	1E				3E	>	6E	>	
1F	US	33			(4)	3F	?	6F	?	

(1) X'15' goes to user buffer
(2) X'0D' goes to user buffer
(3) user buffer contents erased
(4) no character goes to user buffer

Table Q-1. Input Conversion – TTY (cont.)

ASCII		EBCDIC				ASCII		EBCDIC		
Code	Character	Code	Character	EOM	Note	Code	Character	Code	Character	EOM
40	@	7C	@			60		38	CD	
41	A	C1	A			61	a	38	CD	
42	B	C2	B			62	b	38	CD	
43	C	C3	C			63	c	38	CD	
44	D	C4	D			64	d	38	CD	
45	E	C5	E			65	e	38	CD	
46	F	C6	F			66	f	38	CD	
47	G	C7	G			67	g	38	CD	
48	H	C8	H			68	h	38	CD	
49	I	C9	I			69	i	38	CD	
4A	J	D1	J			6A	j	38	CD	
4B	K	D2	K			6B	k	38	CD	
4C	L	D3	L			6C	l	38	CD	
4D	M	D4	M			6D	m	38	CD	
4E	N	D5	N			6E	n	38	CD	
4F	O	D6	O			6F	o	38	CD	
50	P	D7	P			70	p	38	CD	
51	Q	D8	Q			71	q	38	CD	
52	R	D9	R			72	r	38	CD	
53	S	E2	S			73	s	38	CD	
54	T	E3	T			74	t	38	CD	
55	U	E4	U			75	u	38	CD	
56	V	E5	V			76	v	38	CD	
57	W	E6	W			77	w	38	CD	
58	X	E7	X			78	x	38	CD	
59	Y	E8	Y			79	y	38	CD	
5A	Z	E9	Z			7A	z	38	CD	
5B	[[	4F	|			7B	¦¦	38	CD	
5C	\\\	4A	¢			7C	¦¬	38	CD	
5D	]¬[	5F	¬			7D	¦¦	38	CD	
5E	^^↑	6A	^			7E	ESC	3A		(1)
5F	– – ←	6D	–			7F	RUBOUT	32		(2)
	most TTY's CII 70 015 68 ASCII						CII 70 015 68 ASCII			

(1) X'FE' goes to buffer
(2) last character in user buffer erased

Table Q-2. Input Conversion – K/D

ASCII		EBCDIC			
Code	Character	Code	Character	EOM	Comments
0	NULL	36			
1	SOH	37			
2	STX	33		X	Begin message mode text (no echo)
3	ETX	33		X	End message mode text (no echo)
4	EOT	36			
5	ENQ	09			
6	ACK	06			Unlock keyboard
7	BEL	39		X	Cursor up
8	BS	18			
9	HT	39		X	Cursor right
A	NL	39		X	
B	VT	39		X	Cursor home
C	FF	39		X	Hard copy on
D	CR	39		X	Cursor return
E	SO	39		X	Enter normal mode
F	SI	39		X	Enter insert mode
10	DLE	39		X	Hard copy off
11	DC1(XON)	39		X	Roll forward
12	DC2(TAPE)	12			
13	DC3(XOFF)	39		X	Roll backward
14	DC4(TAPE)	12			
15	NAK	39		X	Transmit data
16	SYN	16			
17	ETB	17			
18	CAN	39		X	Erase text
19	EM	39		X	Cursor left
1A	SS	39		X	Cursor down
1B	ESCI	1B			
1C	FS	39		X	Reserved for software FWD
1D	GS	39		X	Turn page BACK
1E	RS	34			
1F	US	1F			
20	blank	40	blank		
21	!	5A	!		
22	"	7F	"		
23	#	7B	#		
24	$	5B	$		
25	%	6C	%		
26	&	50	&		
27	'	7D	'		
28	(	4D	(		
29	)	5D	)		
2A	*	5C	*		
2B	+	4E	+		
2C	,	6B	,		
2D	-	60	-		
2E	.	4B	.		
2F	/	61	/		
30	0	FO	0		
31	1	F1	1		
32	2	F2	2		
33	3	F3	3		
34	4	F4	4		
35	5	F5	5		
36	6	F6	6		

Table Q-2. Input Conversion – K/D (cont.)

ASCII		EBCDIC			
Code	Character	Code	Character	EOM	Comments
37	7	F7	7		
38	8	F8	8		
39	9	F9	9		
3A	:	7A	:		
3B	;	5E	;		
3C	<	4C	<		
3D	=	7E	=		
3E	>	6E	>		
3F	?	6F	?		
40		62			
41	A	C1	A		
42	B	C2	B		
43	C	C3	C		
44	D	C4	D		
45	E	C5	E		
46	F	C6	F		
47	G	C7	G		
48	H	C8	H		
49	I	C9	I		
4A	J	D1	J		
4B	K	D2	K		
4C	L	D3	L		
4D	M	D4	M		
4E	N	D5	N		
4F	O	D6	O		
50	P	D7	P		
51	Q	D8	Q		
52	R	D9	R		
53	S	E2	S		
54	T	E3	T		
55	U	E4	U		
56	V	E5	V		
57	W	E6	W		
58	X	E7	X		
59	Y	E8	Y		
5A	Z	E9	Z		
5B	[	41	[		
5C	~	4A	¢		
5D	]	51	]		
5E	^	6A	^		
5F	—	6D	—		
60	@	7C	@		
61	a	81	a		
62	b	82	b		
63	c	83	c		
64	d	84	d		
65	e	85	e		
66	f	86	f		
67	g	87	g		
68	h	88	h		
69	i	89	i		
6A	j	91	j		
6B	k	92	k		
6C	l	93	l		
6D	m	94	m		

Table Q-2. Input Conversion – K/D (cont.)

ASCII		EBCDIC			
Code	Character	Code	Character	EOM	Comments
6E	n	95	n		
6F	o.	96	o		
70	p	97	p		
71	q	98	q		
72	r	99	r		
73	s	A2	s		
74	t	A3	t		
75	u	A4	u		
76	v	A5	v		
77	w	A6	w		
78	x	A7	x		
79	y	A8	y		
7A	z	A9	z		
7B	{	42	{		
7C	¬	5F	¬		
7D	}	52	}		
7E	|	4F	|		
7F	RUBOUT	FF			

OUTPUT SPECIAL CHARACTER HANDLING ROUTINES

In special cases, the output conversion table contains an F followed by the code number of a special handling routine. Routines and their functions are:

F1 Null – no character is placed in the output buffer.

F2 Cr and Lf are placed in the buffer.

F4 Output message is complete.

Notes:

1. All empty squares in the following tables are coded F1 and no character is sent to the terminal.
2. In each square, the upper character represents the graphic(s) or the name of the character while the lower characters represent the hexadecimal value of the character to be transmitted (but not including the parity which is in the K/D Table).
3. The usable F codes are those with bad parity; F1, 2, 4, 7, 8, B, D, and E; since other F codes represent correct parity-checked characters.

Table Q-3. Output Conversion (EBCDIC – ASCII (TTY))

		Least significant digit (of EBCDIC)															
Most Significant Digit (of EBCDIC)		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	NUL F4	SOH 01	STX 02	ETX 03	EOT 04	HT 09	ACK 06	BEL 07	CAN 18	ENQ 05	NAK 15	VT 0B	FF 0C	CR F2	SO 0E	SI 0F
	1	DLE 10	DC1 11	DC2 12	DC3 13	DC4 14	NL F2	SYN 16	ETB 17	BS 08	EM 19	SS 1A	ESC 1B	FS 1C	GS 1D	RS 1E	US 1F
	2																
	3																
	4	blank 20	[5B	{								~ ¢ \ 5C	2E	< 3C	(28	+ 2B	| 5B
	5	& 26	] 5D	}								! 21	$ 24	* 2A	) 29	; 3B	¬ 5D
	6	- 2D	/ 2F	\ 40								↑^ 5E	, 2C	% 25	← _ 5F	> 3E	? 3F
	7											: 3A	# 23	@ 40	' 27	= 3D	" 22
	8		A 41	B 42	C 43	D 44	E 45	F 46	G 47	H 48	I 49						
	9		J 4A	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52						
	A			S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A						
	B																
	C		A 41	B 42	C 43	D 44	E 45	F 46	G 47	H 48	I 49						
	D		J 4A	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52						
	E			S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A						
	F	0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37	8 38	9 39					ESC ALT 7E	23

Table Q-3. Output Conversion (EBCDIC – ASCII (TTY)) (cont.)

Most Significant Digit (of EBCDIC)	Least Significant Digit (of EBCDIC)															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 00	SOH 01	STX 02	ETX 03	EOT 04	HT 09	ACK 06	BEL 07	EOM 18	ENQ 05	HAK 15	VT 0B	FF 0C	CR 0D	SO 0E	SI 0F
1	DLE 10	DC1 11	DC2 12	DC3 13	DC4 14	NL 0A	SYN 16	ETB 17	BS 08	EM 19	SS 1A	ESC 1B	FS 1C	GS 1D	RS 1E	US 1F
2																
3																
4	blank 20	[5B	{ 7B								~ ¢ \ 5C	. 2E	< 3C	(28	+ 2B	| 7E
5	& 26	] 5D	} 7D								! 21	$ 24	* 2A	) 29	; 3B	¬ 7C
6	- 2D	/ 2F	\ 40								↑ ^ 5E	, 2C	% 25	← _ 5F	> 3E	? 3F
7											: 3A	# 23	@ 60	' 27	= 3D	" 22
8		a 61	b 62	c 63	d 64	e 65	f 66	g 67	h 68	i 69						
9		j 6A	k 6B	l 6C	m 6D	n 6E	o 6F	p 70	q 71	r 72						
A			s 73	t 74	u 75	v 76	w 77	x 78	y 79	z 7A						
B																
C		A 41	B 42	C 43	D 44	E 45	F 46	G 47	H 48	I 49						
D		J 4A	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52						
E			S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A						
F	0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37	8 38	9 39						# 23

APPENDIX R. MEMORY ALLOCATION

Figure R-1 shows how core storage areas are allocated in the Monitor system.

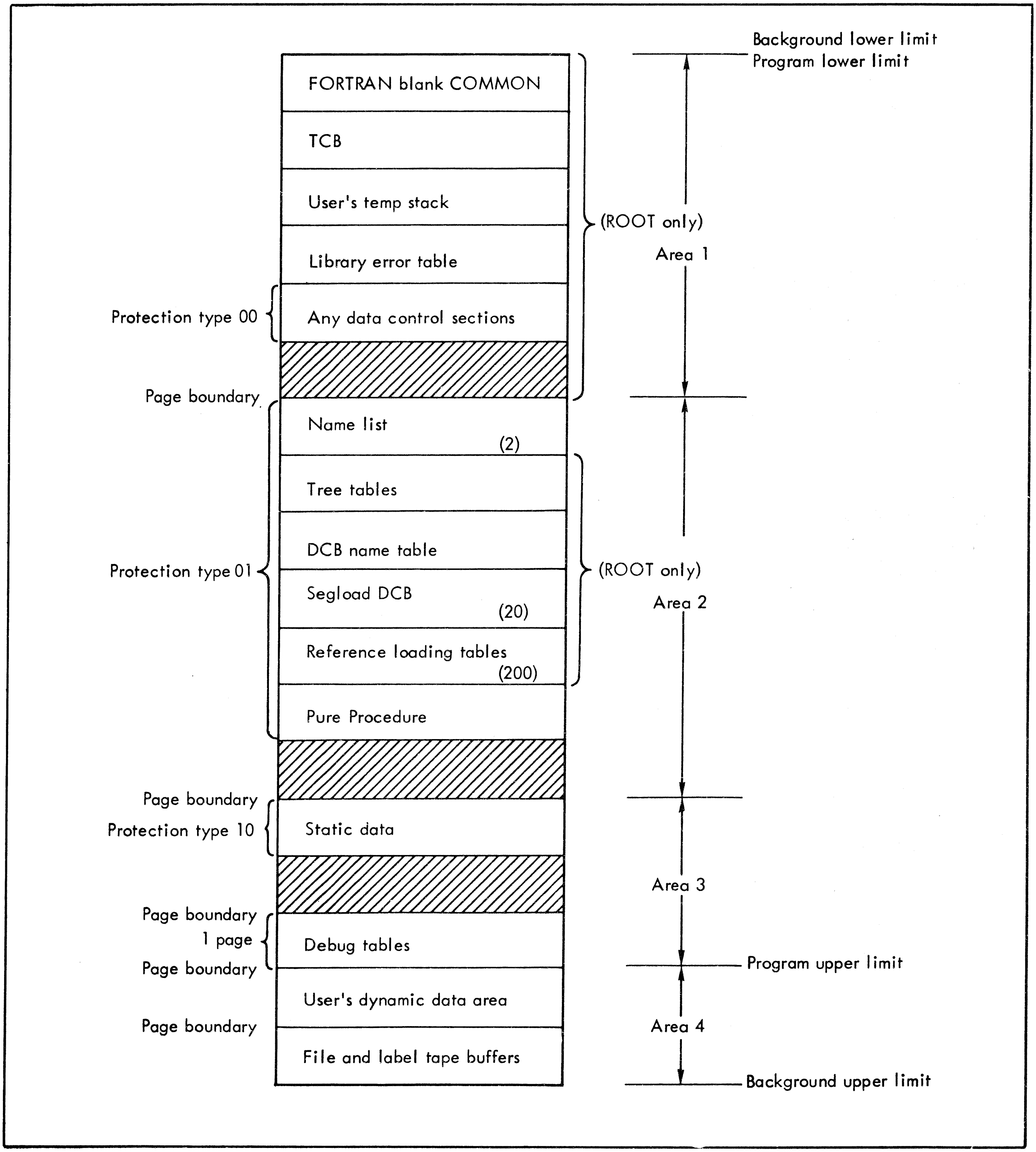


Figure R-1. Memory Allocation

APPENDIX S. MONITOR MODULES LISTED BY CATALOG NUMBER

704020 M:DISPLAY

This module is used to display specified Monitor information on the OC device as requested by an unsolicited key-in.

704023 M:COPYFILE

This module copies opened output files to the checkpoint device. Input files are also copied if the IN option has been specified.

704031 BLOCK/DEBLOCK

The blocking/deblocking routines are designed to supplement the Monitor for a given set of special packing/reading operations on sequential files.

704032 SENSE SWITCH SIMULATION

Three routines are provided to allow for setting, resetting, and testing the pseudo sense switches.

704058 M:CKDCB

This module contains the DCB for checkpoint use.

704363 FLOATING POINT INSTRUCTION SIMULATOR

This routine simulates the following instructions: FAS, FAL, FSS, FSL, FMS, FML, FDS, and FDL.

704364 DECIMAL INSTRUCTION SIMULATOR

This routine simulates the following instructions: DL, DST, DA, DS, DM, DD, DSA, DC, PACK, UNPK and EBS.

704365 BYTE-STRING INSTRUCTION SIMULATOR

This routine simulates the following instructions: MBS, CBS, TBS, and TTBS.

704366 CONVERT INSTRUCTION SIMULATOR

This routine simulates the instructions CVA and CVS.

704368 I/O HANDLER INTERFACE

All handlers require this interface to initiate data transfer and service device interrupts.

704369 MAGNETIC TAPE I/O HANDLER

This handler performs all magnetic tape I/O. The I/O HANDLER INTERFACE (Catalog No. 704368) is used and must be present.

704370 LINE PRINTER I/O HANDLER

This handler performs all line printer output. The I/O HANDLER INTERFACE (Catalog No. 704368) is used and must be present.

704371 CARD READER I/O HANDLER

This handler performs all card reader input. The I/O HANDLER INTERFACE (Catalog No. 704368) is used and must be present.

704372 CARD PUNCH I/O HANDLER

This handler performs all card punch output. The I/O HANDLER INTERFACE (Catalog No. 704368) is used and must be present.

704373 TYPEWRITER I/O HANDLER

This handler performs all typewriter I/O. The I/O HANDLER INTERFACE (Catalog No. 704368) is used and must be present.

704374 PAPER TAPE I/O HANDLER

This handler performs all paper tape I/O. The I/O HANDLER INTERFACE (Catalog No. 704368) is used and must be present.

704708 M:RMESS

This module handles communication for the various overlay segments in the Monitor.

704709 F:CFUD

This module contains DEFs for the various entries in the Current File Use (CFU) table of the Monitor.

704710 M:OPN

This module is called to handle the M:OPEN CAL.

704711 M:OBSE

This module performs security evaluation, file name checking and scanning the File Parameter Table when a DCB is to be opened.

704712 M:OPNL

This module is called by M:OPN (Catalog No. 704710) to open DCBs assigned to labeled tape.

704713 M:RDF

This module handles the reading of records from disc files.

704714 M:TYPR

This module handles the M:TYPE, M:PRINT, and M:KEYIN CALs. It also is called to type tape messages such as !!MOUNT, !!SCRATCH, !!SAVE, and !!DISMOUNT.

704715 M:IORT

This module receives M:READ and M:WRITE CALs. It interprets the parameter list and goes to the appropriate Read or Write routine. It handles the reading of the C device by checking for control commands. M:CHECK and M:MERC CALs are also handled by this module.

704716 M:IOD

This module processes all of the M:DEVICE CALs and the M:SETDCB CAL.

704717 M:LBLT

This module handles the writing of labeled tape, it also performs M:CVOL functions for labeled and unlabeled tapes. It is called by M:CLS, Catalog No. 704722, to close DCBs assigned to labeled or unlabeled tape.

704718 M:RDL

This module handles the reading of labeled tape. If the end-of-volume sentinel is encountered, the routine switches to the next volume and reprocesses the Read request.

704719 M:WRTD

This module handles the writing of records for DCBs assigned to output devices (except disc files and labeled tape).

704720 M:POS

This module handles M:WEOF, M:REW, M:PFIL, and M:PRECORD CALs.

704721 M:WRTF

This module handles the writing of records to disc files, including updating old records and writing new ones. It also processes the M:DELETE CAL.

704722 M:CLS

This module closes DCBs. It waits for the completion of outstanding I/O associated with the DCB. If the DCB is assigned to tape, M:LBLT (Catalog No. 704717) is called. On other devices, an EOD is punched (if appropriate) and the routine exits.

704723 M:DLT

This module handles the deletion of records after M:WRTF (Catalog No. 704721) locates the record to be deleted.

704724 M:ROOT

This module of the overlay loader controls segment loading and the calling of loader passes.

704725 M:INIT1

This module initializes the loader's temp stack, allocates memory for M:PASS1, and reads the LOAD control command table.

704726 M:PASS1

This module passes through the object modules, making a table of all external REFs and DEFs, control sections, dummy sections, and forward references, and a table of expressions defining them. It also performs the library search.

704727 M:INIT2

This module initializes M:PASS2 data and allocates memory for M:PASS2.

704728 M:PASS2

This module controls the sequencing and loading of the second pass segments. It also reads the tables built by MPASS1.

704729 M:ALLOCATE

This module assigns memory locations to all control sections and dummy sections.

704730 M:EVLOAD

This module reads all object and library load modules, and forms the memory image of the program and its relocation dictionary.

704731 M:WRITESEG

This module writes the load module segments and builds the necessary tables for the root of the load module. It also contains the logic for the MODIFY control command.

704732 M:CCI

This module reads each control command, interprets the first field of the command, and passes control to the appropriate subroutine to process the command.

704733 M:DCBs

This module provides a complete set of Monitor DCBs for the control command processor.

704734 M:ASSGR

This module processes the ASSIGN control command.

704735 M:CHARROUT

This module provides a set of routines to perform the syntax checking and analysis of the control commands for the control command processor.

704736 M:MDLIMIT

This module contains the standard job limits.

704737 M:DEBUG

This module processes the INCL, PMD, PMDI, SNAP, SNAPC, IF, AND, OR, and COUNT control commands.

704738 M:LOADR

This module processes the LOAD control command.

704739 M:TAPEFCN

This program implements the control commands REW, WEOF, and PFIL.

704740 M:TPECHST

This module contains character scan routines for use with TAPEFCN and FILEMNGE.

704741 M:TREER

This module processes the TREE control command.

704742 M:TELSCPE

This module packs the LOAD control command table, tree table, and the relocatable object module table (ROMT) into one table and sorts the ROMT by segment.

704743 M:RUNR

This module processes the RUN control command.

704744 M:LDPRGM

This module calls the program loader to load user or processor programs, sets memory protection on the program loaded, allocates the specified number of file and index buffers, and passes control to the loaded program.

704745 M:EXIT

This module processes the M:EXIT, M:ERR, and M:XXX calls.

704746 M:KEYIN

This module process all unsolicited key-ins.

704747 M:FILEMNGE

This module implements the FMGE control command.

704748 M:SITB

This module contains the tables used for Monitor overlay loading.

704749 M:SIMPLEDBUG

This module is an optional simple debug package.

704750 M:TABLES

This module contains all common constants and data used by the Monitor.

704751 M:IOQUEUE

This module accepts all I/O requests, stores them into a queue, and dispatches them to the appropriate I/O handler. In addition, it handles all direct operator communication.

704752 M:ENTRY

This module contains common routines for all trap and interrupt entry and exit functions.

704753 M:MONSEGLD

This module controls Monitor internal segment loading.

704754 M:CLOCKI

This module receives the clock interrupt signal and updates the time and date. Any active timers are also checked and updated.

704755 M:TOPRT

This module defines all segment numbers and entry points for the nonsymbiont version.

704756 M:SNAP

This module processes debug procedures M:SNAP, M:SNAPC, M:IF, M:AND, M:OR, and M:COUNT. The desired snaps are given on the DO device.

704756 M:DUMP

This module accepts requests to display the contents of registers or memory on the DO device.

704758 M:CALPROC

This module decodes all CAL1, 1 entries to the Monitor and gives control to the proper routines.

704759 M:MTRAP

This module implements user service procedures M:TRAP, M:STRAP, M:TRTN, M:INT, M:STIMER, and M:TTIMER.

704760 M:TIME

This module gets the current time of day and date and returns them in printable form.

704761 M:CHKPT

This module checkpoints the specified program and all specified files.

704762 M:IOREC

This module performs the recovery specified on direct device key-ins.

704763 M:SEGLD

This module reads in segments requested by M:SEGLD procedures or implicitly by reference loading.

704764 M:AVR

This module is called by IOQUEUE to recognize labeled tapes (automatic volume recognition).

704765 M:ALTCP

This module decodes all CAL1 service requests except CAL1, 1.

704766 M:BUFGRAN

This module performs the following functions:

1. Obtains and/or releases symbiont-cooperative core buffers, symbiont-cooperative file context buffers, Monitor buffers, file index buffers, and file blocking buffers.

2. Obtains and/or releases symbiont-cooperative disc granules, foreground disc granules, and background disc granules.

704767 M:MEMALOC

This module of subroutines performs the following functions:

1. Gets or frees dynamic program data pages.

2. Gets or frees program COMMON pages.

3. Gets the address limits of COMMON pages.

4. Sets memory protection for core page(s).

704773 M:DISCIO

This module is the disc handler.

704776 M:PRGMLDR

This module reads a load module, relocates it, if necessary, builds any necessary debug tables, and merges ASSIGN information with the load module's DCBs.

704851 7-TRACK MAGNETIC TAPE I/O HANDLER

This handler provides the use of all 7-track options in conjunction with the MAGNETIC TAPE I/O HANDLE, (Catalog No. 704369). In addition, it provides a simulated read-reverse capability.

704852 FORTRAN BCD CONVERSION SUBROUTINE

The subroutine provides conversion from FORTRAN BCD to EBCDIC and EBCDIC to FORTRAN BCD. It is used in conjunction with card reader, card punch, and 7-track tape handlers.

704866 M:OPNT

This module opens DCBs assigned to unlabeled magnetic tape.

704867 M:PASS1SG

This module accepts control cards that will allow the selection and updating of elements on the master installation tape. A new installation tape can be created if specified. Control commands accepted are SELECT, UPDATE, and SYSWRT.

704868 M:MONITOR2

This module accepts the :MONITOR control command via card input and uses the parameters specified to generate the load module M:CPU. If some parameters are not specified, default values are used.

704869 M:FORMLIB2

This module uses the output from the DLIB module and generates the inputs to the loader program. A load module is generated which contains all ROMs that make up the public library. The loader is called again to form many load modules that make up the system library.

704870 M:DLIB2

This module builds a list of ROM names to be passed to the FORMLIB module. The ROM names are taken from the :DLIB card, and the list contains ROM names that will make up the public and system libraries.

704871 M:RESERVE

This module accepts data from the RESERVE control card to generate load modules M:FCOM and M:RESF.

704872 M:RJITGEN

This module is called on by either M:RESERVE or M:SYSINT2 to form two job information tables. The tables are written to disc under the load module names M:RJIT and M:NRJIT.

704873 M:DEFDCBS

This module contains the DCBs required for the DEF processor of System Generation.

704874 M:FMGEDCBS

This module contains the DCBs used by FILEMNGE.

704875 M:DEF

This module writes a boot record and a bootable version of the Monitor formed by M:PASS2.

704876 M:TMTOPO

This module works in conjunction with M:DEF to form a tape from the system generated by M:ASS2. It copies all modules from the current account to PO.

704877 M:WRITEMON

This module is used with M:PASS1 to write a boot record and a bootable System Generation system to tape.

704878 M:BITOTM

This module is used to initialize a system. It reads all load modules from BI and transfers them to the system account.

704879 M:IOQS

This module is similar to Catalog No. 704751, except that it also handles symbiont communication.

704880 M:TOPRTS

This module defines all of the segment numbers and entry points used for overlaying the Monitor.

704881 M:TFILE

This module puts the desired file name into the IDT file, to be released at the end of the job.

704882 M:DUMINIT

This module is included to provide a DEF to delimit the area of the CLS segment dedicated to initialization.

704883 M:DUMMYCCL

This module is effectively a dummy that replaces the co-op CCLOSE in a nonsymbiont system.

704884 M:TOP

This module provides an upper delimiter for the initialization area of the CLS segment.

704885 M:TABLESS

This module contains all common data and tables for the symbiont version of the Monitor.

704886 M:ENTRYS

This module contains all standard entry and exit logic for trap and interrupt processing.

704887 M:SYSCCI0

This module contains the pass-0 control command interpreter vector. The control commands accepted are :GENCHN, :GENOP, :GENDC, :GENDCB, :GENSZ, :GENMD, :GENDEF, :GENEXP, and :GENDICT.

704888 M:SYSPHA0

This module contains the processors that syntactically process the control commands :GENCHN, :GENOP, :GENSZ, :GENDC, and :GENDCB.

704889 M:SYSPHB0

This module contains the processors that syntactically process the control commands :GENMD, :GENDEF, :GENEXP, and :GENDICT.

704890 M:SYSPHC0

This module copies all of the load modules from the M:BI file to the M:TM file. It then performs an update of these load modules with the designated control commands (:GENMD, :GENDEF, :GENEXP, and :GENDICT) as directives.

704891 M:SYSCHR0

This module performs a requested character string retrieval and analysis. The character string source is found in the specified control command buffer.

704892 M:SYSDCB0

This module defines the DCBs required by System Generation pass-0.

704893 M:SYSSDEV2

This module processes any SDEVICE control commands and will generate the load module M:SDEV.

704894 M:LNKLDTRC

This module determines whether a CAL is for M:LINK or M:LDTRC. For an M:LINK CAL, the calling routine and all of its necessary information are swapped out by the M:CK DCB. For both M:LINK and M:LDTRC CALs, the called overlay segment is loaded and entered at its entry point. In some cases an M:LDTRC CAL may result in the return to a previously swapped-out routine.

704895 M:CKPTDCBM

This routine is used to pack the DCBs that are open when a checkpoint is requested. Information concerning the DCBs is also saved.

704896 M:SYSCCI2

This module contains the pass-2 character routines and a vector that identifies a control command type and then loads the appropriate overlay segment.

704897 M:SYSDVLB2

This module processes the DEVICE and STDLB control commands and generates the load module IOTABLE.

704898 M:SYSMOD

This module performs the actual modification to a load module element as specified by a MODIFY control command directive.

704899 M:SYSCCLD2

This module sets up a call to load an overlay segment for the second pass of System Generation.

704900 M:SYSLBLD2

This module calls CCLOAD to load either the DLIB or FORMLIB processors.

704901 M:CCITBLS

This module contains the constants and tables used by the control command processor.

704902 M:CC1SUBR

This module contains a set of closed subroutines to be used by the control command processor.

704903 M:CCILIST

This module lists all control commands and error messages for the control command processor.

704904 M:CCIOPNF

This module is used to open a specified Monitor DCB to a specified file.

704905 M:JOBR

This module processes JOB and FIN control commands.

704906 M:LIMR

This module processes the LIMIT, STDLB, MESSAGE, TITLE, POOL, and SWITCH control commands.

704907 M:WASGPL

This module writes the information from the ASSIGN control commands into a system file on disc.

704908 M:READBI

This module reads the binary information from the ASSIGNed BI device when the BI option is specified on the LOAD control command, and writes the information into the system BI file.

704909 M:ENDJOB

This module performs the end-of-job functions of releasing all temporary system files and temporary user files, displays the accounting information, and intializes the Job Information Table for the next job.

704910 M:TYPABORT

This module lists the abort messages on the operator's console and the listing log device.

704911 M:C

This module defines the Data Control Block (DCB) for the control command input.

704912 M:OC

This module defines the Data Control Block (DCB) for the operator's console.

704913 M:LO

This module defines the Data Control Block (DCB) for the listing output.

704914 M:LL

This module defines the Data Control Block (DCB) for the listing log.

704915 M:DO

This module defines the Data Control Block (DCB) for the diagnostic output.

704916 M:PO

This module defines the Data Control Block (DCB) for the punch output.

704917 M:BO

This module defines the Data Control Block (DCB) for the binary output.

704918 M:LI

This module defines the Data Control Block (DCB) for the library input.

704919 M:SI

This module defines the Data Control Block (DCB) for the source input.

704920 M:BI

This module defines the Data Control Block (DCB) for the binary input.

704921 M:SL

This module defines the Data Control Block (DCB) for the system log.

704922 M:SO

This module defines the Data Control Block (DCB) for the source output.

704923 M:CI

This module defines the Data Control Block (DCB) for the compressed input.

704924 M:CO

This module defines the Data Control Block (DCB) for the compressed output.

704925 M:AL

This module defines the Data Control Block (DCB) for the accounting log.

704926 M:EI

This module defines the Data Control Block (DCB) for the element input.

704927 M:EO

This module defines the Data Control Block (DCB) for the element output.

704928 M:SYMSUBR

This module is composed of a group of small subroutines that perform computational functions common to more than one subroutine in symbiont-cooperative control.

704929 M:SYMCOM

This subroutine module allows the operator to communicate with any symbiont via a key-in.

704930 M:SACT

This subroutine module loads from disc and/or initiates or reactivates symbionts queued by device in the SYMTAB table by SYMBOM or by any other means (symbiont queued itself up, etc.).

704931 M:SUSPTERM

This module allows a symbiont to suspend or terminate operation on the appropriate device and types an appropriate message on the console typewriter.

704932 M:COOPRES

This module determines whether a peripheral device I/O request should be queued directly (nonsymbiont device request) or indirectly, via M:COPNRES (symbiont device request).

704933 M:COPNRES

This module obtains card images written by the card reader symbiont, or buffers card images, printer images, or typewriter images for the output symbiont.

704934 M:SYMCR

This module is the card reader symbiont.

704935 M:SYMPPRTY

This module is the output symbiont.

704936 M:ADDF

This module adds a specified input symbiont file or output cooperative file to the file directory (SYMFILE).

704937 M:GETF

This module is used to obtain the proper input device file for the input cooperative or the proper output device file for the output symbiont.

704938 M:REQDC

This module performs the following functions:

1. Obtains a disc granule for a requesting input symbiont.
2. Enables an output symbiont to release a disc granule at I/O end action.
3. Obtains a core buffer for a requesting symbiont.
4. Enables a symbiont to release a core buffer at I/O end action.

704939 M:DCBPASS1

This module is used by M:PASS1 and consists of DCB tables.

704940 M:SYSDCB2

This module provides the DCB tables for the use by the M:PASS2 processor.

704941 M:GO

This module defines the Data Control Block (DCB) for GO-file use.

704957 M:DLIMIT2

This module builds the load module M:DLIMIT containing system default limits generated from information taken from the :DLIMIT control command.

704958 M:SYSINT2

This module processes the INTS and INTR control commands and generates load modules M:INT and M:INTLOC.

704959 M:SGLDR

This is a modified version of Catalog No. 704724 for use with System Generation. It controls segment loading and the calling of loader passes.

704963 M:DELPRI

This module accepts unsolicited key-ins to delete files or change the priority of files in the symbiont file directory.

704964 M:SRCHF

This module is used to search for and change item keys in the symbiont file directory.

704966 M:ARDL

This module controls key Read and Read Reverse operations on labeled tape.

704967 M:CLOCK

This module accepts data from the :TIME control card. Two tables are generated, one contains the names taken from the card and the other has zeroed data cells. The tables are written to disc under the load module name M:TIME.

704968 M:LNKIO

This module forms a DCB in the Monitor temp stack, opens, reads or writes, and closes a file when the M:LINK or M:LDTRC procedure is executed.

704969 M:INTRT

This module handles all external interrupts connected centrally to a foreground task and restores the Monitor overlay environment when a resident foreground task exits.

704970 M:LOADRT

This module loads a foreground task and connects it to an external interrupt.

704971 M:EXITF

This module handles all normal and abnormal exits from a foreground task.

704972 M:FORCAL

This module processes the foreground call M:TRIGGER, M:DISABLE, M:ENABLE, M:DISARM, M:ARM, M:DCAL, M:CAL, M:SLAVE, M:MASTER, M:TERM, M:SBACK, M:RBACK, M:SXC, and M:RXC.

704973 M:LNKRT

This module performs the I/O rundown for M:LINK and releases all programs swapped by M:LINK calls.

APPENDIX T. LOADER CONSTRUCTED DCB'S AND SYSTEM DCB'S

The overlay loader constructs a DCB corresponding to all external REFs with names beginning either with "F:" or "M:". These 48-word DCBs contain the default assignment to a device, an operational label, a default record size, and a default function, if the DCB name is recognized. These DCBs also contain space for a 3-word file name, an account number, a password, 3 input serial numbers and 3 output serial numbers. If the user requires a DCB with room for additional information, he may either construct the DCB himself with the M:DCB procedure or request it explicitly from the system library by including the DCB name as an element file with account no. ":SYS" in his !LOAD, !OVERLAY, or !OLAY card.

The recognized DCB names and their defaults are listed below.

DCB Name	Function	Record Byte Size	Operational Label
M:C	Input	120	C
M:OC	Input/output	85	OC
M:LO	Output	132	LO
M:LL	Output	132	LL
M:DO	Output	132	DO
M:PO	Output	80	PO
M:BO	Output	120	BO
M:LI	Input	120	LI
M:SI	Input	80	SI
M:BI	Input	120	BI
M:SL	Output	132	SL
M:SO	Output	80	SO
M:CI	Input	120	CI
M:CO	Output	120	CO
M:AL	Output	80	AL
M:EI	Input	120	EI
M:EO	Output	120	EO
M:GO	Output	120	NO
F:101	Input	0	OC
F:102	Output	0	OC
F:103	Input	0	PR
F:104	Output	0	PP
F:105	Input	80	SI
F:106	Output	120	BO
F:108	Output	132	LO

Only the DCBs whose names begin with "M:" are in the system library.

INDEX

CAL1-TO-FUNCTION INDEX

Printed in France

COMPAGNIE INTERNATIONALE POUR L'INFORMATIQUE 68 route de Versailles 78 – LOUVECIENNES Tél. 951-86-00
S.A. au Capital de 81.000.000 F
R.C. Versailles 66 B 756 – INSEE 283.78.350.0.005

COMPAGNIE INTERNATIONALE POUR L'INFORMATIQUE
68 route de Versailles 78 - Louveciennes - Téléphone : 951 86-00

J.M.L. RICHARD