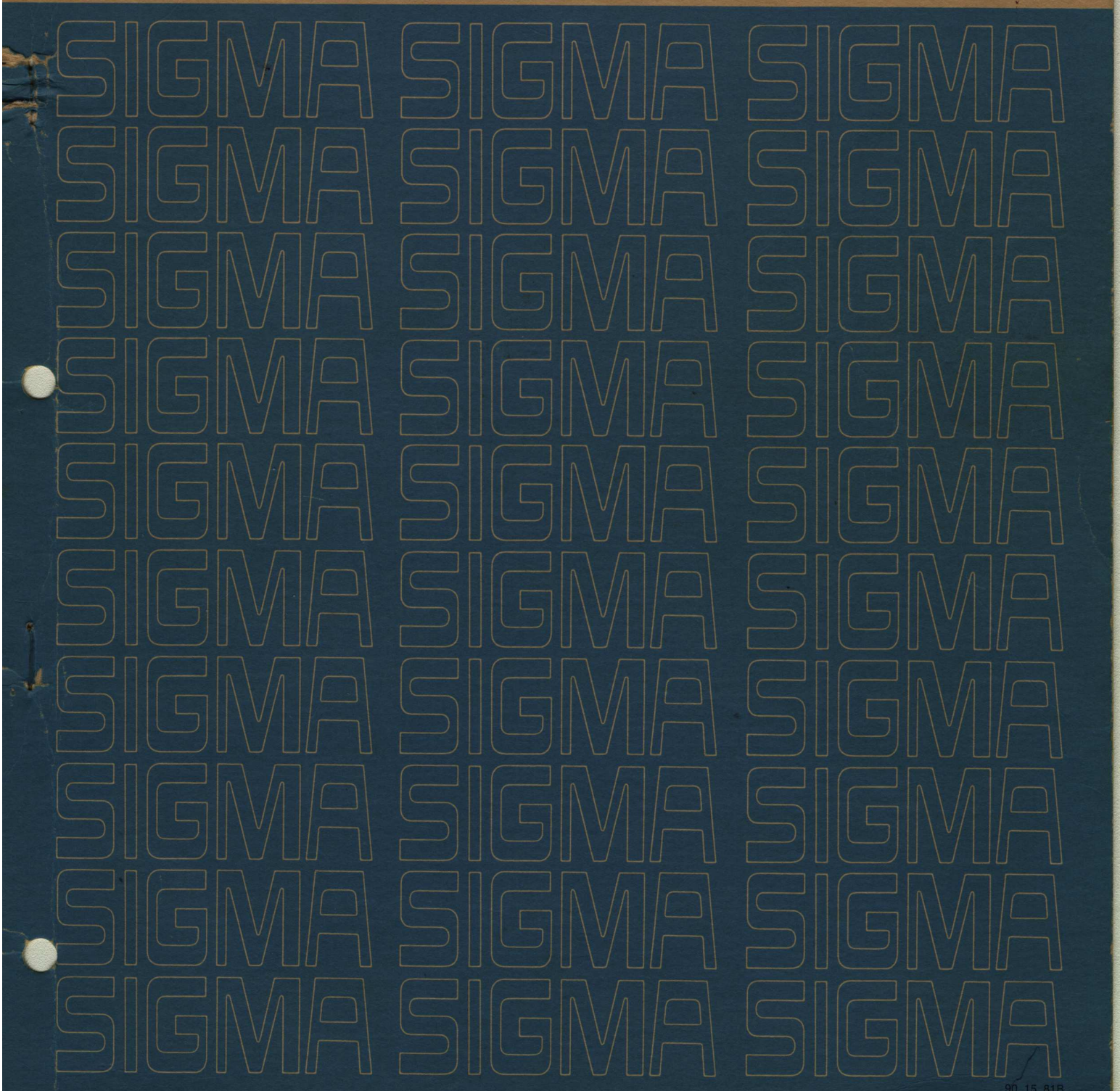
EPB

XDS

Xerox Data Systems

XDS SIGMA 5/7 REAL-TIME BATCH MONITOR (RBM)

Reference Manual



90 15 81B

CAL1-TO-FUNCTION INDEX

Call	FPT Code	Function	Comment	Page
CAL1,1	X'29'	CHECK		37
	X'15'	CLOSE		37
	X'22'	DEVICE/FILE		42
	X'05'	DEVICE FORMAT		42
	X'14'	OPEN		36
	X'1C'	PFIL		41
	X'1D'	PREC		41
	X'10'	READ		38
	X'01'	REW		40
	X'02'	UNLOAD		40
	X'03'	WEOF		40
	X'11'	WRITE		39
CAL1,2	X'01'	PRINT		41
	X'02'	TYPE		41
CAL1,5	X'04'	ARM		52
	X'04'	CONNECT		51
	X'01'	DISABLE		52
	X'03'	DISARM		52
	X'02'	ENABLE		52
	X'12'	IOEX (SIO)		43
	X'13'	IOEX (TIO)		43
	X'14'	IOEX (TDV)		43
	X'15'	IOEX (HIO)		43
	X'08'	MASTER		52
	X'0B'	RLS		50
	X'0C'	RUN		49
	X'07'	SLAVE		52
	X'10'	STOP SYS. I/O		42
	X'11'	START SYS. I/O		42
	X'0E'	STOP BG. I/O		42
	X'0F'	START BG. I/O		42
	X'00'	TRIGGER		52
CAL1,8	X'01'	SEGLOAD		52
	X'10'	TIME		53
	X'14'	TRAP		50
CAL1,9 1		EXIT	Address value shown in Call column; FPT not used.	51
CAL1,9 3		ABORT		51
CAL1,9 5		TRTN		51
CAL1,9 9		WAIT		53

Price: $5.50

REAL-TIME BATCH MONITOR (RBM) REFERENCE MANUAL

for

XDS SIGMA 5/7 COMPUTERS

90 15 81B

April 1970

XDS

Xerox Data Systems/701 South Aviation Boulevard/El Segundo, California 90245

Printed in U.S.A.

REVISION

This publication, XDS 90 15 81B, is a revision of the XDS Sigma 5/7 Real-Time Batch Monitor Reference Manual, XDS 90 15 81A (dated November 1969). The manual has been slightly reformatted for faster referencing, and new examples of various processes have been incorporated. This revision also documents the Sigma 5/7 Real-Time Batch Monitor Extensions, Group 1. 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 No.
XDS Sigma 5 Computer Reference Manual	90 09 59
XDS Sigma 7 Computer Reference Manual	90 09 50
XDS Sigma 5/7 Real-Time Batch Processing Operations Manual	90 16 47
XDS Sigma 5/7 Real-Time Batch Monitor Technical Manual	90 17 00
XDS Sigma Real-Time Batch Monitor User's Guide	90 16 47
XDS Sigma 5/7 Mathematical Routines Technical Manual	90 09 06
XDS Sigma 5/7 Symbol and Meta-Symbol Reference Manual	90 09 52
XDS Sigma 5/7 Macro-Symbol Reference Manual	90 15 78
XDS Sigma 5/7 SL-1 Reference Manual	90 16 76
XDS Sigma 5/7 Extended FORTRAN IV-H Reference Manual	90 09 66
XDS Sigma 5/7 Extended FORTRAN IV-H Operations Manual	90 11 44
XDS Sigma 5/7 Extended FORTRAN IV-H Library/Run-Time Technical Manual	90 11 38
XDS Glossary of Computer Terminology	90 09 57

CONTENTS

APPENDIXES

ILLUSTRATIONS

TABLES

DEFINITION OF TERMS

active foreground program: a foreground program is active if it is resident in memory, connected to interrupts, or in the process of being entered into the system via a !RUN control command.

addend value: a hexadecimal or decimal 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 to a halfword, word, or doubleword address.

background area: that area of core storage allocated to batch processing. This area may be checkpointed for use by foreground programs.

background program: any program executed under Monitor control in the background area with no external interrupts active. These programs are entered through the batch processing input stream.

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

centrally connected interrupt: an interrupt that is connected to a Monitor interrupt routine which first saves the environment of the system and then switches the environment to that of the task that gets control when the interrupt occurs.

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.

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

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

declaration: an object language 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 external name associated with a particular object language declaration.

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

directly connected interrupt: an interrupt which, when it occurs, causes control to go directly to the task. e.g., execution of the XPSD instruction in the interrupt location gives control to the task rather than first going to a Monitor interrupt routine.

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).

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

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.

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

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).

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.

foreground area: that portion of memory dedicated specifically for foreground programs.

foreground program: a load module that contains one or more foreground tasks.

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

foreground task: a body of procedural code that is associated with (connected to) a particular interrupt and that is executed when the interrupt occurs.

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 RAD file of Relocatable Object Modules formed by a processor.

granule: a block of disc sectors containing a specified number of words.

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 a !C key-in.

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

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.

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 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.

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

logical device: a peripheral device that is represented in a program by an operational label (e.g., BI or BO) 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 and control commands.

object language: the standard binary language in which the output of a compiler or assembler is expressed.

object module: the series of records containing the load information pertaining to a single program or subprogram. Object modules serve as input to the Overlay 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.

OV file: is a temporary RAD file that contains an executable program formed by the Overlay Loader if a program file name was not specified at load time. Used primarily to test new programs or new versions of programs.

Overlay Loader: a processor that loads and links elements of overlay programs.

overlay program: a segmented program in which the segment currently being executed may overlay the core storage area occupied by a previously executed segment.

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.

Relocatable Object Module: a program, or subprogram, generated by a processor such as Macro-Symbol, FORTRAN, etc. (in XDS Sigma 5/7 object language).

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

ROM: Relocatable Object Module.

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., RAD).

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 EBCDIC 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.

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 included in a program being created.

Task Control Block (TCB): a table of program control information built by the relocating loader when a load module is formed. The TCB is part of the load module and contains a temp stack and the data required to allow reentry of library routines during program execution. The TCB is program associated and not task associated.

Temp Stack: a push-down stack created by the Overlay Loader and used by the Monitor and System Library routines.

1. INTRODUCTION

OPERATING SYSTEM

The Sigma 5/7 Real-Time Batch Monitor (RBM) is the major control element in an installation's operation system. Operating in a real-time environment, the Monitor provides for concurrent background/foreground processing with emphasis on foreground operations.

The operating system consists of the Monitor, language translators, service programs, batch (background) user's programs, and real-time (foreground) user's programs. In general, the Monitor governs the order in which these programs are executed and provides services common to all of them.

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

As the requirements of an installation increase, the operating system can be enlarged, modified, or updated. The ability to adapt to new requirements is inherent in the system design. Once a system is generated, it can quickly be expanded to include users' programs, data, and system libraries.

A user's program and data may be temporarily incorporated in the operating system or 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 XDS Sigma 5/7 Real-Time Batch Monitor Operations Manual.

RBM TERMS AND PROCESSES

The following items are either unique to the RBM system or have specific meaning within the RBM context. Terms and processes not defined below are fully explained in the appropriate chapter.

TASK

A task is a body of foreground procedural code associated with a specific interrupt.

PROGRAM

A program is a body of procedural code and data that is identifiable by name. A program is created at load time from object modules and exists after load time in core image form. A program is identified by a name so that it may be loaded or released on request. Background programs are loaded by control commands; foreground programs can be loaded or released upon request through operator key-in, control command, or system call from a foreground task.

FOREGROUND

The foreground is the set of all tasks in the system that are currently connected to external interrupts. The priority level and activation sequence of each interrupt controls the execution order of the tasks. Foreground tasks are guaranteed memory protection from background processes.

BACKGROUND

The background is the set of all programs that use up any available CPU time after the real-time interrupts are satisfied. In contrast to foreground tasks, background programs are executed serially and their sequence is controlled by control commands.

TEMP STACK

The Temp Stack is a "push-down/pull-up" stack of memory locations allocated by the Overlay Loader. It is used for dynamic temporary storage when Monitor functions and FORTRAN IV-H Library subroutines are called, and is also available as temporary storage for the user.

DATA CONTROL BLOCK

A DCB is a table located in the calling program that contains information used by RBM in the performance of an I/O operation. 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 and the type of operation to be performed.

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.

There are both system DCBs and user-created DCBs. The system DCBs need only be coded as external references in a System Processor or user program; the Overlay Loader will satisfy these external references at load time by furnishing a copy of the appropriate DCBs in the program's root. If a user is not satisfied with the standard DCB parameters furnished by the Overlay Loader, the system DCBs can be coded into the user program's root and the DCB name declared as an external definition.

FUNCTION PARAMETER TABLE

An FPT is a table through which a program communicates with a Monitor function.

TASK CONTROL BLOCK

A TCB is a table containing task-associated parameters. The space for this table is allocated by the user and the entries are used and maintained by the Monitor.

PROGRAM CONTROL BLOCK

The PCB is a table containing program-associated parameters. The PCB is constructed by the Overlay Loader at load time.

CHECKPOINT

Checkpoint is the function of saving a copy of the background program on secondary storage.

RESTART

Restart is the function of restoring a background program from its checkpoint image and resuming its operation from the point of interruption.

REENTRANT SUBROUTINE

A reentrant subroutine can be called by several different tasks. During execution of such a subroutine, a higher priority task can interrupt and call the same subroutine. When the higher priority task has completed execution, control is returned to the subroutine at the interrupted point. Since a reentrant subroutine does not perform any instruction modification and uses the Temp Stack for scratch storage, processing continues as though the subroutine had never been reentered.

PHILOSOPHY OF OPERATION

The Monitor provides for two levels of operation:

1. Real-time foreground processing.
2. Batch processing.

REAL-TIME PROCESSING

Real-time processing, the most critical aspect of multiusage, involves reacting to external events (including clock pulses) within microseconds.

Real-Time programs can be either automatically loaded every time the system is booted from the RAD or loaded and initiated as needed. The first method is used when the real-time process normally remains unchanged and is constantly operative. The second approach is used when real-time operations are executed periodically or irregularly, as in an experimental laboratory.

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

During SYSGEN, a user can reserve a portion of his foreground area for communication between real-time programs. Locations in this area are called foreground mailboxes. The start of this area can be referenced through the system label FP:MBOX. Upon encountering an external reference to FP:MBOX, the Overlay Loader will satisfy the reference with the first location in the mailbox area.

The Monitor provides foreground programs with the facility for direct I/O operations (so-called IOEX operations), wherein the user furnishes the basic hardware commands and does the necessary error checking and recovery. This type of I/O operation provides decreased overhead and greater flexibility as compared to indirect I/O operations.

Foreground programs can be loaded for execution from a background job stack by operator key-in or through a system call by a foreground program, providing the program to be loaded is already on the RAD in core image format. Foreground programs are responsible for initializing the interrupt system and connecting tasks to interrupts. Foreground tasks can be processed compatibly and concurrently with a background production job stack.

RBM will not borrow CPU time from a higher priority task to process I/O requests of a lower priority.

BATCH PROCESSING

The system is capable of processing a continuous series of background jobs with little or no operator intervention. Reducing the need for operator participation ensures faster throughput, makes operations less subject to error, and allows the operator to perform other tasks.

MONITOR

The Monitor controls and coordinates batch (background) and real-time (foreground) processing. Efficient operation is assured by minimizing Monitor overhead in response to an interrupt, and by preserving the relative priority of tasks.

Reentrant service functions perform I/O and control the interrupt system; other service functions load and connect foreground tasks, and control the partition of memory between foreground and background areas.

The Monitor provides for dynamic partitioning of core memory into background and foreground areas under user control.

Parts of the Monitor must remain resident to ensure continuous coordinated operation. Other parts are brought into core memory from secondary storage as required to perform specific functions. Secondary storage management is essential for the Monitor. It is used for system storage to overlay portions of the Monitor, thus minimizing core memory residency. Processing programs are retrieved from the RAD and they too can capitalize on overlay techniques to minimize core memory requirements.

Scratch storage for service programs, processors, and user programs is available on the RAD. In addition, the secondary storage accommodates permanent user files. Permanent user files on disc are provided through the RAD Editor, which can allocate files on the RAD in addition to providing media conversion, RAD mapping (listing of files), and other services.

The Monitor provides foreground programs with a background checkpoint feature that writes the background program on

the RAD (after the I/O requests outstanding at the time of the checkpoint requests have completed) and marks the background core as foreground. The checkpoint of the background is performed implicitly when a request is made to load a foreground program and some portion of the memory occupied by the program lies in the background area. The restart of the checkpointed background is also performed implicitly upon release of the last foreground program using a portion of the memory area required by the checkpointed program.

Monitor features are summarized as follows:

- Efficient I/O service to user programs.
- Dynamic real-time process initiation and execution.
- Utilization and management of rapid-access secondary storage (RAD).
- Comprehensive control of system operation by computer operator.
- Sophisticated (but easy-to-use) processor services for program creation and execution such as FORTRAN IV-H and Macro-Symbol.
- Checkpoint service.
- Job accounting.
- Flexible job scheduling of foreground programs for efficient throughput operation and recognition of installation priorities.
- Modular, flexible design for user modification.
- Use of overlay techniques and Public Library capability to minimize core memory residency.
- Blocking and compression of RAD files.
- Foreground priority scheduling.
- Direct I/O operations.
- Memory protection of the operating environment and real-time processes (except for read protection).

RAD UTILIZATION

A rapid access disc (RAD) is essential for efficient operation of the Monitor. It minimizes core memory requirements by storing all Monitor, processor, and user program overlays, yet provides very fast access whenever an overlay is called into memory.

RAD FILES

During System Generation, the RAD(s) is divided into large blocks called RAD areas. These areas represent functional groupings of RAD files. All RAD files within an area are either temporary or permanent, and have the same software write protection. Since the entire RBM system is RAD oriented, every job will directly or indirectly involve the use, modification, allocation, or release of RAD files. Listed below are the RAD areas and the types of files that occupy them.

- Foreground Programs area (FP) contains a collection of foreground programs and optional User Libraries and Public Libraries. Reference library routines are included in the user load modules at "load time". Public Libraries are a group of routines shared by a number of programs and are called into core for execution only when referenced.
- System Programs area (SP) contains the Monitor and the set of language translators used by the local installation, such as Macro-Symbol and FORTRAN IV-H. The area also contains the System Library (i.e., FORTRAN IV-H Library/Run-time), RAD Editor, and Overlay Loader. All translators are called by user jobs to execute in the background core space.
- Background Programs area (BP) contains the set of user operational programs that execute in the background. ~~and a background User Library if desired.~~
- Data areas (D1 through DF) are divided into foreground and background data areas, and are used for storing data. Foreground programs cannot write into files in background data areas or vice versa; however, either type of program can freely read from both areas.
- Background Temp area (BT) contains temporary (scratch) files (X1 through Xn where n is a SYSGEN parameter) used by background programs for intermediate storage in processing. Their use is identical to scratch tapes on magnetic tape units. (Note that temp files are destroyed at the end of each job step unless ~~a :SAVE command is present within a job step~~ the SAVE option or an !ALLOCT command is used.) Temp files are automatically destroyed when a new !JOB command is encountered in a job stack regardless of :SAVE commands, and there is no way to save data on temp files from one job to another.

The GO and OV files are also temporary files in the BT area but have more permanence than the Xi temp files. The GO file contains Relocatable Object Modules (ROMs) formed by a processor if the GO option was specified on a control command. It is used by FORTRAN IV-H, Macro-Symbol, and Symbol programs for assemble-and-go type of operations. The OV file contains the executable program formed by the Overlay Loader if a program file name was not specified at load time. It is the default output file for the Overlay Loader, and is used primarily to test a new program that has no permanent file defined, or to test a new version of a program without destroying the current version. A program in the OV file is always named OV, and is called for execution via an !ROV control command.

Note that both GO and OV are used for communication between job steps but not jobs. There is no Monitor protection for these files between one job and another.

JOB ACCOUNTING

The Monitor provides accounting services for user job activity on the Sigma computer. Because of the system's multi-usage capability, the accounting information can indicate total elapsed time or actual machine time of each job.

Background job accounting is an option selected at SYSGEN. To correctly calculate the elapsed time of a background job, all foreground tasks need to be centrally connected (see "Connecting Real-Time Tasks to Interrupts" in Chapter 4). Otherwise, the foreground task's execution time will be included in the elapsed time of the background job.

To prevent including foreground execution time with background elapsed time, foreground response time to an interrupt will be slightly slower (5 microseconds). However, at SYSGEN, the user has the option to include foreground execution time with background elapsed time to prevent this degradation of foreground response time.

At the beginning of a background job, the date and start time (in hours and minutes) will be logged on the LL device. At the end of a background job, the total time of the job (in hours, minutes, and seconds) will also be logged on LL. This information plus the account number and user name is then written on the "AL" file in the Background Data area of the RAD. The "AL" file must be defined via the RAD Editor; it must be in the D1 area of the RAD, and allocated a minimum size of 256 words. This file can be purged periodically by the operator.

The total time of a job is computed from the time the !JOB command is read until the next !JOB or !FIN command is encountered. Following the !FIN command and until the next !JOB command, all unused time is charged to the idle account.

To attach a date and time to each job, the user is required to input the date and time of day (to the nearest minute) whenever the system is booted into core.

PUBLIC LIBRARY

If an RBM system has several programs that share a group of subroutines, this set of subroutines can be collected in core in a "public library". This preselected set is loaded by the Overlay Loader into a previously defined file on the RAD. The Loader also writes the names and entry points of all the routines into the same RAD file. Then, whenever the Overlay Loader loads a foreground or background program that references one of the "public" routines, it links up the appropriate branch to the Public Library copy instead of loading a separate copy. This can represent a considerable saving in space for a large system.

If the appropriate Public Library was not already present in core, it will be automatically loaded into its specified foreground location whenever a program is loaded that uses it. Routines in the Public Library will execute under the same WRITE key as the calling program; therefore, a Public Library routine used by both foreground and background must only store into the calling program area.

RBM CONTROL TASK

The RBM Control Task will perform the following functions:

1. "I/O cleanup" and "I/O start" when any of these functions are deferred from the I/O interrupt task because of priority considerations.
2. Loading and initialization of foreground programs and loading of background programs.
3. Release of foreground and background programs.
4. Console interrupt and operator key-in processing.
5. Postmortem dumps for the background.
6. Checkpoint and restart of background.

The RBM Control Task is connected to the lowest priority interrupt in the system at System Generation time. For configurations without system interrupts, the Control Task is connected to the console interrupt.

OVERLAYS

RBM is overlayed to minimize core residence requirements. The overlays consist of all Control Task subtasks such as the various key-in processors.

Foreground functions and frequent-use functions are all resident.

MEMORY PROTECTION

The Monitor provides memory protection for all input operations, except direct input and write protection for RAD files. The hardware write-lock feature furnishes memory protection for all non-I/O operations.

The hardware write-lock feature inhibits both foreground programs (write-lock 10) and background programs (write-lock 01) from storing outside their own memory area. An exception is the Public Library which resides in the foreground area of memory but executes under the key of the calling program (either background or foreground). This permits Public Library routines to use a Temp Stack in the calling routine's portion of memory. Also, some Monitor routines are given a skeleton key. One such routine is the Job Control Processor, which executes in the background, but has to set system flags in the Monitor portion (write-lock 11) of memory.

Background programs causing protection violations are aborted; foreground programs are not aborted but must process all traps resulting from memory violation (see TRAPS system call).

Memory protection on all input operations performed via Monitor functions (except IOEX) is guaranteed by the Monitor software. The Monitor checks the validity of the input area on all read operations to ensure that the area is wholly contained in the calling program's write-lock area. If an

attempt is made to read into an invalid area of core, an error condition is returned to the error address specified by the user. If no address is specified, the job is aborted. An "FG" key-in is required before a foreground program can be loaded from the background job stack; this protects the foreground from an error in a background job stack.

RAD WRITE PROTECTION

The Monitor furnishes software write-protection of RAD files above and beyond that furnished by the hardware write-protect switches on the RAD. Background programs are allowed to write only in the Background Data area or Background Temp areas of the RAD. Foreground programs are allowed to write only in the Foreground Data areas. The foreground user is responsible for ensuring that IOEX (direct I/O) writes only in the IOEX Access area of the RAD. The Systems program area, Foreground program area, and Background program area can be written into only if the "SY" key-in is in effect.

Similarly, the Overlay Loader and RAD Editor (background processors) are allowed to write in nonbackground areas only if the "SY" key-in is in effect. Any RAD write-protection violation will result in a write-protection error indication return, and the write order will not be carried out.

All RAD files (and all of memory) can be read by a user without restrictions. There is no Monitor-furnished read protection for memory or RAD files.

PROCESSING PROGRAMS

The following language translators are available for inclusion in the operating system:

FORTRAN IV-H

SL-1

Symbol

Macro-Symbol

FORTRAN IV-H is a compiler that operates in the background but is capable of generating code that will function in a real-time environment.

SL-1 is a simulation language to solve differential equations as the fundamental procedure in simulating parallel, continuous systems. An extensive set of macros permit the user to simulate a wide variety of linear and nonlinear elements through the use of single-operator statements. These prototype statements are inserted into the user program each time a macro is referenced by name.

Symbol is a one-pass assembler that accepts symbolic input and outputs programs in Sigma 5/7 standard object language.

Macro-Symbol is a two-pass assembler with procedure capability that accepts both symbolic and compressed format programs as input, and provides standard sequential editing for compressed input files. The assembler outputs programs in Sigma 5/7 standard object language.

SERVICE PROGRAMS

Service Programs provide routines for performing frequently used functions. The service programs include the Overlay Loader and the RAD Editor.

OVERLAY LOADER

The Overlay Loader (a background processor) can be used to create overlay programs for later execution in either the foreground or background. Thus, if a foreground program can tolerate a slight delay in reading the overlays into core for execution, either foreground or background programs of virtually unlimited size can be constructed even though core size is restricted. For example, a 1400-word overlay can be input in about 50 milliseconds, assuming a Model 7204 RAD is available. That is, the time required to bring in an overlay for execution is the time of the one RAD access required to read the overlay. Since a program is stored on the RAD in core image format, it can be loaded very quickly as one logical record per segment. A program loaded by the Overlay Loader can be entered permanently into the System Programs Directory, Foreground Programs Directory, or Background Programs Directory, or it can be loaded on a temporary file in the Background Temp area of the RAD.

The overlay structure as illustrated in Figure 1 is restricted to a permanently resident root section and any number of overlay segments. A blank COMMON and labeled COMMON data area can be established for use by the root and overlay segements. Each segment is created by the Loader from one or more object modules output by the Symbol, Macro-Symbol, or FORTRAN IV-H processors. The Loader will build the Program Control Block, the OVLOAD table (used to load the overlay segments at execution time), allocate or build DCBs, and allocate the temp stack. It will also load library modules to satisfy unsatisfied references encountered in the loading process. A maximum of two libraries can be searched. Library search and loading are extremely fast, due to special tables that are added to the library files at the time the library is created on the RAD.

The overlay segments must be explicitly defined at load time and explicitly called at execution time. There is no provision for implicitly calling in an overlay segment. All segments in a path may communicate with each other via REF/DEF linkages, but it is the user's responsibility to ensure that any segment referenced is currently in core.

RAD EDITOR

The RAD Editor (a background processor) controls RAD allocation for areas containing permanent RAD files and performs utility functions for all areas. The RAD areas with permanent files include Background Programs, Foreground Programs, System Programs, and data areas.

The RAD Editor performs the following functions:

1. Adds or deletes entries to the permanent file directories.
2. Compacts the RAD areas by relocating RAD files and updating/compacting directories to regain space within an area.
3. Maps permanent RAD file allocation.

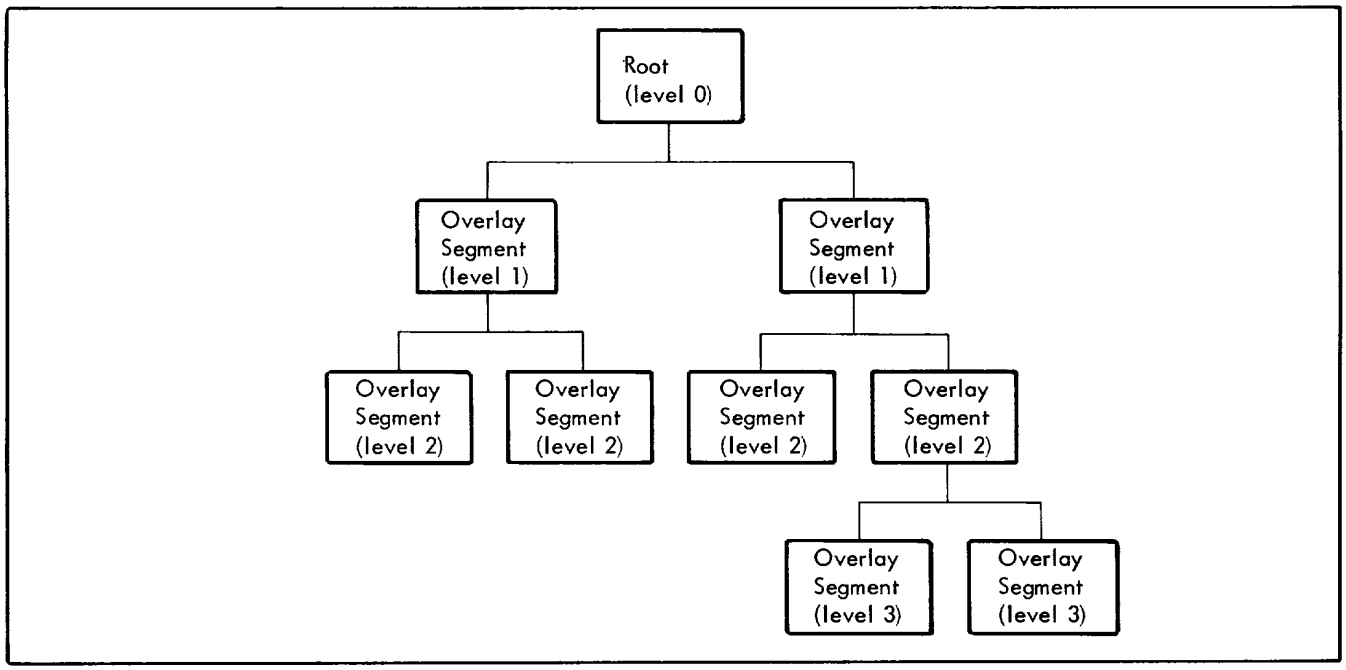


Figure 1. Overlay Structure

4. Builds and maintains library files on the RAD for use by the Overlay Loader.
5. Copies permanent RAD files from one file to another.
6. Saves the contents of RAD areas in self-reloadable form.
7. Restores RAD areas previously saved.
8. Dumps the contents of permanent RAD files or areas.

JOB ORGANIZATION

The user controls the construction and execution of a background job by means of control cards placed before, within, and following the input card decks. These control cards, interpreted by the Job Control Processor, Overlay Loader, or RAD Editor, 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 (postmortem dump) requirements.

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 results in the execution of a processing program such as a language translator, service program, or user's program.

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 routine saves all data needed to restart a checkpointed job along with the job.

HARDWARE CONFIGURATIONS

The minimum configuration required and supported by RBM for either a Sigma 5 or Sigma 7 is the following:

Sigma 5 or Sigma 7 CPU with two clocks and Integral IOP

Memory Protect Feature[†]

Memory Module: 4096 words[††]

Memory Increment: 4096 words each for a total of 12,228 words[††]

Keyboard/Printer with Paper Tape Reader/Punch

RAD Control and .75MB RAD Storage Unit

Interrupt Control Chassis[†]

Priority Interrupt, Two Levels[†]

External Interface Feature[†]

Note that an external interrupt level is required at execution time for each real-time task in the system, and another external interrupt level is required for the RBM Control Task.

[†]To run background only, these items are not required in the minimum configuration. To run background/foreground, the complete list is required.

[††]Memory Module and Memory Increment comprise a total of 16K.

In addition to the previous list, any items from the list below can be added for increased performance and will be specifically supported by RBM. Other items can be added to this list but will not receive any special RBM support.

Floating-Point Arithmetic

Memory Module

Memory Increment

Multiplexor IOP with Eight Channels

Additional Eight Multiplexor Channels

Selector IOP

Decimal Arithmetic

Keyboard/Printer

Paper Tape Reader/Punch (High-Speed)

Card Readers

Card Punches

RADs

9-Track Magnetic Tape

7-Track Magnetic Tape

BCD and Binary Packing Options for 7-Track Magnetic Tape

Buffered Line Printers

Plotters

SYSTEM CONFIGURATIONS

CORE SPACE CONSIDERATIONS FOR A MINIMUM SYSTEM

The minimum size of a resident RBM is 5-1/2K words. This will support foreground/background processing under the minimum hardware configuration, but not floating-point or decimal arithmetic simulation software, or job accounting. The floating-point simulation software requires about 500 words; decimal arithmetic requires about 800 words; and job accounting requires about 100 words.

RBM will support the following system processors in a maximum background space of 11K (some of which require considerably less than 11K):

Macro-Symbol

Symbol

FORTRAN IV-H

SL-1

Overlay Loader

RAD Editor

WIRING OF EXTERNAL INTERRUPTS

External interrupts must be wired so that their priority and address both have a corresponding monotonically increasing or decreasing sequence. That is, the highest priority interrupt must be connected to the lowest address interrupt cell; the next highest priority interrupt must be connected to an address greater than the highest priority interrupt, etc.

A user can wire external interrupts to give them a higher priority than the I/O interrupt within the following restrictions:

1. A task connected to the high-priority interrupts cannot use any Monitor function that performs I/O (e.g., READ, WRITE, IOEX).

2. Such a task cannot perform its own I/O if the I/O device is connected to the central I/O interrupt level.

In general, a foreground task should not be connected to an interrupt whose priority is higher than the I/O interrupt except in a situation where instant response must be guaranteed. In this special case, a task that must perform I/O could do direct data I/O or trigger a lower priority task to perform system I/O.

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 Macro-Symbol, Symbol, FORTRAN IV-H SL-1, Overlay Loader, and RAD Editor processors, the operator, and the peripheral equipment.

Control commands have the general form

```
! mnemonic specification
```

where

! is the first character of the record and identifies the beginning of a control message.

mnemonic is the mnemonic code name of a control function or the name of a processor. The name may begin any number of spaces after the ! character, except for an EOD command.

specification is a listing of required or optional specifications. This may include keywords, labels, and numeric values appropriate to the specific command.

In this manual, the options that may be included in the specification field of a given type of control command are shown enclosed in brackets (no brackets are actually used in control commands, and parentheses are required to indicate the grouping of subfields).

One or more blanks must separate the mnemonic and specification fields, but no blanks can be embedded within a field. A control command is terminated by the first blank after the specification field, or, if the specification field is absent and a comment follows the command, the command is terminated by a period after the blank that follows the mnemonic field. Annotational comments detailing the specific purpose of a command may be written following the command terminator, but no control command record can contain more than 80 characters.

A control command can be continued from one record to the next by using a semicolon to replace the comma as a subfield terminator in the specification field of the command. Column one of the continuation card must contain either an exclamation mark (if the control command is read by the Job Control Processor), or a colon (if the command is read by the Overlay Loader or RAD Editor). See the control command examples given later in this chapter for an illustration of the proper use of the semicolon.

Communication between the operator and the Monitor is accomplished via control commands, key-ins, and messages.

Control commands are usually input to the Monitor via punched cards; however, any input device(s) may be designated for this function. All control commands are listed on the output device designated as the listing log (normally a line printer). In this manner, the Monitor keeps the operator informed regarding the progress of the job. When a job is aborted, all control commands skipped over until the next JOB command is encountered are listed on LL with a greater than character (>) in column one.

Note that in all control commands, the first three characters after the exclamation character are sufficient to define any mnemonic code or keyword.

Control commands may be categorized as follows:

System Control	Debug Control
JOB	PMD
ASSIGN	
LOAD	Utility Control
ATTEND	PFIL
MESSAGE	PREC
PAUSE	SFIL
CC	REWIND
LIMIT	UNLOAD
STDLB	WEOF
ROV	DAL
RUN	
POOL	Processor Control
ALLOBT	
MODIFY (special)	OLOAD
	RADEDIT
Input Control	MACRSYM
	SYMBOL
EOD	FORTRANH
FIN	SL-1

JOB CONTROL PROCESSOR

All control commands are read from the "C" (op label) device by the Job Control Processor (JCP). The JCP is a special processor loaded into the background by RBM upon the initial "C" key-in. The JCP is also reloaded into the background following each job step within a job. A job step is defined as all control commands required for the setup and execution of a single processor or user program within a job stack.

The JCP processes each control command until a request is made to execute a processor or user program, at which time the appropriate program is read into the background and given control. A detailed description of the JCP interface with the system processors or user programs is given later in this chapter under "Processor Control Commands".

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. The JOB command causes the temporary assignments of all operational labels (except the "C" operational label) to be reset to their permanent assignments.

The form of the JOB command is

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

where

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

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

Note that a comma separates the optional subfields. The account number must precede the name, and both fields must be present if either one is present.

Example:

```
! JOB 12345,JOBSAMP1
```

The above example defines the account number for the job as 12345, and the user as JOBSAMP1.

ASSIGN The ASSIGN control command specifies the physical peripheral devices or RAD files to be used in processing the current job step, and the uses to which they will be put. ASSIGN commands must appear prior to the appropriate RUN or Processor name command and affect only that one job step. Each ASSIGN command assigns a Data Control Block (DCB) name to an operational label (logical device name), a RAD file, or a physical device. An operational label is a symbolic name used to identify a logical system device (see Table 1). The "name" to which a DCB is assigned may be either a system 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.

or an operational label, background temp file, permanent RAD file, or numeric zero.

If there is an error in an ASSIGN command, the entire command must be input again.

Table 1. Monitor Operational Labels

Label	Reference	Comments
BI	Binary input	Used to input binary information.
CI	Compressed input	Used by Macro-Symbol.
SI	Symbolic input	Used to input source (symbolic) information.
C	Control command input	Used by the Monitor and processors to read control commands.
BO	Binary output	Used to output binary information.
DO	Diagnostic output	Used by the Monitor for postmortem dump and diagnostic messages.
LO	Listing output	Used for object listings from assemblies and compilation.
CO	Compressed output	Used by Macro-Symbol.
LL	Listing log	Used by the Monitor to log control commands and other system messages.
OC	Operator's console	Used by the Monitor for key-ins and operator control. (Always assigned to a keyboard/printer.)
SO	Symbolic output	Used by SL-1.
PL	Plotter output	Used by user programs

Table 2. I/O Device Type Codes

yy	Device Type
TY	Typewriter
LP	Line printer
CR	Card reader
CP	Card punch
9T	9-track magnetic tape
7T	7-track magnetic tape
PP	Paper tape punch
PR	Paper tape reader
DC	RAD or other disc storage
PL	Plotter
NO	Not a standard device. Used as a special purpose device for IOEX.

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 (single devices)	Refers to a device number (00 through 7F).
80 ≤ dd ≤ FF (multiple devices)	Refers to a device controller number (8 through F) followed by a device number (0 through F).

The form of the ASSIGN control command is

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

where

dcb is the name (not exceeding eight characters in length) of the DCB to be assigned. It must be the first subfield following ASSIGN, and must be followed by a name specification (see below). The first two characters of a user's DCB name must be "F:" (e.g., F:PRINT or F:BI). The first two characters of a system DCB name are "M:" (e.g., M:LO).

area specifies the RAD area if the DCB is to be assigned to a RAD file, and must be one of the codes itemized in Table 5.

name specifies a system physical device name, a system operational label, a background temp file name (X1-X9, GO, OV), a permanent RAD file name, or a numeric zero. In the "0" case, no output will be generated by the DCB. If assignment is to a permanent RAD file, and if "name" is omitted, the DCB is assigned to the entire RAD area.

Table 5. RAD Area Mnemonics

Code	Mnemonic
SP	Systems Program area
FP	Foreground Program area
BP	Background Program area
BT	Background Temp area
XA	IOEX area
CK	Checkpoint area
D1 : DA : DF	Data area (number of data areas are defined at SYSGEN)
Note: If the DCB is assigned to a background temp file, the area can be omitted.	

The options below are used only if the user creates the DCB or changes some of the DCB's parameters. Note that DCB parameters not specified on the ASSIGN command are not changed from their initial value. The initial values of the DCB parameters depend upon how the DCB was created. Parameters of System DCBs have standard default values. DCBs allocated by the Overlay Loader (F:DCBs) are set to all zeros. User created DCBs have the initial values specified by the user.

Mode may be any or all of the following:

{BCD} specifies the EBCDIC or automatic device mode.

{BIN} specifies the binary device mode.

{VFC} specifies vertical format control.

{NOVFC} no vertical format control.

{PACK} specifies that the packed binary or unpacked binary mode is to be used for 7-track magnetic tape.

{UNPACK}

number of recovery tries

TRIES,value specifies the maximum number of recovery tries to be attempted for an I/O operation. The value must be less than 256.

default record length

RECL,value specifies the default record length in bytes. The value n must be 1 ≤ n ≤ 32,767. This record length is used for all requests referencing the DCBs that do not explicitly specify a record length in the FPT.

Examples:

1. Assign listable output to a magnetic tape:

```
! ASSIGN (M:LO, 9TA81), VFC
```

This example assigns the M:LO DCB to a 9-track magnetic tape. Vertical format control is also specified, so the first byte in each record is a format control byte for the line printer.

2. Assign binary output to the GO file on RAD:

```
! ASSIGN (M:BO,GO)
```

This example assigns the M:BO DCB to the GO file. Note that in this case the RAD area can be omitted.

3. Assign source input to a RAD file in the D1 Data area:

```
! ASSIGN (M:SI, D1, PRESTORE)
```

This example assigns the M:SI DCB to the RAD file PRESTORE, which is in the D1 area. This type of assignment could be used to assemble a source program that had been prestored onto a RAD file.

4. Build a user DCB that was left empty at load time:

```
! ASSIGN (F:XX, 7TAE0), PACK, (TRIES, 3),
  (RECL, 80)
```

This example builds a user DCB, F:XX, and also assigns F:XX to a 7-track magnetic tape. The packed binary mode (PACK) will be used in accessing the tape, and a maximum of three recovery tries (TRIES, 3) will be attempted for a possible tape parity error. The default record size to be read or written is 80 bytes (RECL, 80).

5. Assign a user DCB to read nonstandard binary codes:

```
! ASSIGN (F:INP, CRA03), BIN
```

This example assigns the user DCB, F:INP, to the card reader, and specifies that the binary mode is to be used in reading the cards. This type of assignment would be used to change an existing DCB to read nonstandard binary cards.

LOAD The LOAD control command directs the JCP Loader to load a program on the RAD and absolutize it for its core execution location.

The form of the LOAD command is

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

where the options are

IN[, area], name specifies the input device as a system physical device name, a system operational label, or a RAD file from which the object modules will be loaded. The default input device is the one assigned to the BI operational label.

OUT[, area], name specifies the output device as an operational label or a RAD file on which the loaded program is written. The default output device is the OV file. A foreground program can only be loaded on the FP area of the RAD or the OV file.

EXLOC, value specifies the execution location (in hexadecimal) of the program being loaded. The default location will be the start of background.

SEG, value specifies the decimal number of overlay segments that follow the root. The default value is zero, which means only a root is being loaded.

MAP specifies that a map of the loaded program be output to the LO device. The default is no map.

{F / B} specifies that the program being loaded is a foreground program (F) or background program (B). The default is a background program.

The primary function of the JCP Loader is to load the Overlay Loader and RAD Editor at SYSGEN time. However, the JCP Loader will load any nonoverlaid program on the RAD under certain restrictions (see Appendix D).

Example·

Load Overlay Loader from cards:

```
! LOAD (IN,CRA03),(OUT,SP,OLOAD),
  (SEG,5),MAP
```

This command would be used to load the Overlay Loader onto its permanent file (OLOAD) on the SP area of the RAD. Five overlay segements (SEG,5) are specified and a MAP of the load is requested. The complete deck structure required to perform the load is illustrated in Figure 2.

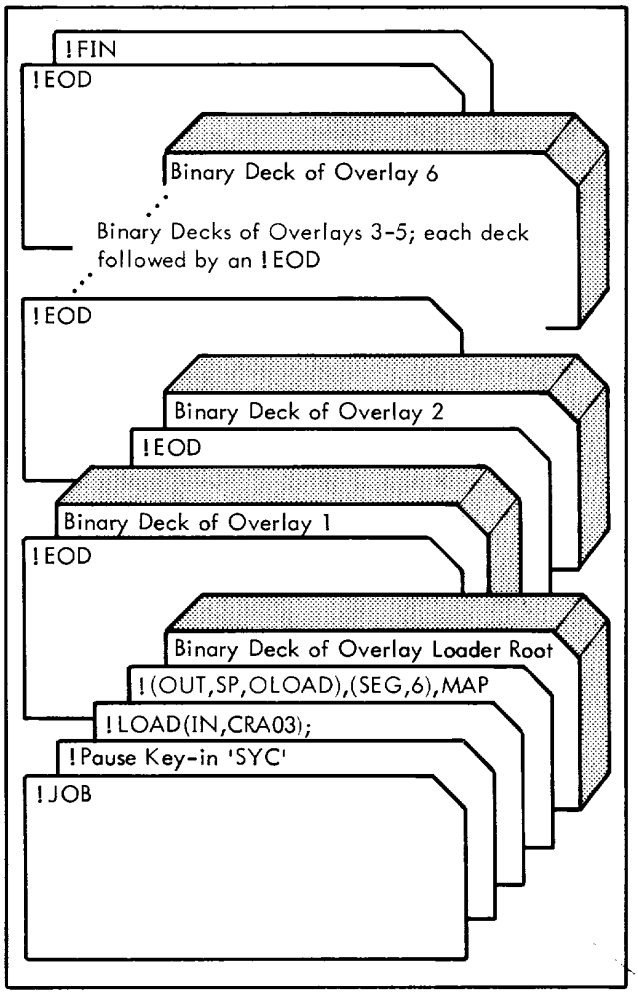


Figure 2. Loading Overlay Loader from Cards

ATTEND The ATTEND control command is used during an attended run and indicates that RBM is to go into a WAIT condition after a WAIT system call, or after an abort from the background. After an unsolicited key-in of "C", background processing will continue from the point of the wait. If the ATTEND control command is not specified, and an abort or error condition occurs, or if a WAIT system call is made, the Monitor does not pause for operator intervention but skips all control commands, binary records, and data until a JOB or FIN command is encountered. When in skipping mode, all control commands encountered will be listed on the LL device, with a greater than character (>) replacing the exclamation mark in column one. Hence, the default mode of operation (no ATTEND command) is for closed-shop batch processing, with no halts between jobs after an abort.

The form of the ATTEND command is

```
! ATTEND
```

The effect of an ATTEND command exists for one job only. Normally, the ATTEND command immediately follows the JOB command.

MESSAGE The MESSAGE control command is used to type a message to the operator. The message will be typed on the OC device, and normal processing will continue after the message is output.

The form of the MESSAGE command is

```
! MESSAGE message
```

where message is any comment to the operator, up to a full card image (80 columns). The message 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.

Note that the entire card image, including the !MESSAGE, will be output to LL and OC.

Example:

```
! MESSAGE SEND ALL SAVE TAPES TO JOHN SMITH
```

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

```
!!MESSAGE SEND ALL SAVE TAPES TO JOHN SMITH
```

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

PAUSE The PAUSE control command is similar to the MESSAGE command except that the JCP will enter a WAIT state after the message is output to OC to give the operator time to carry out the instructions in the message. Processing is continued after an unsolicited key-in of "C".

The form of the PAUSE command is

```
! PAUSE message
```

where message is any comment to the operator, up to a full card image (80 columns).

Example:

```
! PAUSE KEYIN SYC
```

The above example would cause the Monitor to pause execution with the following message output on LL and OC,

```
!! PAUSE KEYIN SYC
```

giving the operator time to key in SYC, which would permit the user to override the write protection on the RAD and continue the background job.

CC The CC control command removes typewriter override of the C device (see TY key-in description). The next control command will be read from the C device instead of the typewriter.

The form of the CC control command is

```
! CC
```

The CC control command has the same effect as the CC key-in, and can be used whenever the JCP has control.

LIMIT The LIMIT control command is used to set a maximum allowable execution time for a background program. If the job exceeds the time limit, the background is aborted with a postmortem dump (if the dump option was specified via a PMD control command).

The form of the LIMIT control command is

```
! LIMIT n
```

where n specifies the maximum allowable execution time in minutes.

STDLB The STDLB command is used to change the temporary assignment of an operational label, with the exception of OC (operator's console). The operational labels being changed receive the new temporary assignments which stay in effect until the next JOB command is encountered.

The form of the STDLB command is

```
! STDLB (label[,area],name)[,(label[,area],name)...]
```

where

label specifies one of the standard operational labels input during SYSGEN (see Table 2).

area specifies a RAD area for an operational label assignment to a RAD file. If an operational label is assigned to a file in the Background Temp area (X1-X9, GO, OV), only the file name need be specified.

name specifies a physical device name to which the operational label is to be temporarily assigned, a RAD file name, a numeric zero, or another operational label. In the latter case, the first operational label will receive the same assignment as the temporary assignment of the second operational label. If "0" is specified, there is to be no temporary assignment, which means that no output will occur on that label. The C op label cannot be assigned to zero via this command. Note that if an error occurs on a !STDLB command, all fields up to the one in error will be processed.

Example:

Change temporary assignments of operational labels:

```
! STDLB (BO,GO),(CO,D2,COMPRESS),(LO,9TA80)
```

This example could be used for Macro-Symbol assembly to change the binary output to the GO file in the RAD, the compressed output to the COMPRESS file in the D2 area of the RAD, and the listable output to a 9-track magnetic tape.

ROV The ROV command (RUN OV) causes execution of the program (either foreground or background) on the OV file.

The form of the ROV command is

```
! ROV
```

The loading of any program into the foreground area via a ROV control command must be preceded by an FG key-in (see Chapter 3). A foreground program loaded by !ROV is given the name OV. There may be only one such program resident at any time.

RUN The RUN control command causes the named program (either foreground or background) to be executed.

The form of the RUN command is

```
! RUN area,file name
```

where

area,file name specifies a RAD area and the file name of the program in that area that is to be executed. The area must be either SP, FP, or BP. The loading of any program into the foreground area via a RUN control command must be proceded by an FG key-in.

POOL The POOL control command is used to override the default allocation of blocking buffers for the background by the JCP. The POOL command takes effect only for a job step and not for the duration of an entire job.

The form of the POOL command is

```
! POOL n
```

where

n specifies the number of 256-word blocking buffers that are to be allocated to the background for the blocking and deblocking of blocked or compressed RAD files. The n value must be less than 255, and the value of 256n + n + 1 cannot exceed the available background space.

ALLOBT The ALLOBT control command is used to define the files in the BT area of the RAD, and overrides any JCP default definitions. The files input on the ALLOBT command will receive the specified sizes and formats. The files defined via an ALLOBT command will stay in effect only for the current job step unless the SAVE option is invoked. If the SAVE option is used, the ALLOBT command will stay in effect for the entire job (any input for the GO or OV files will always stay in effect for the entire job).

The form of the ALLOBT command is

```
! ALLOBT (FILE,nn)[,(option),(option)...]
```

where

FILE,nn specifies the name of the background temp file to be allocated. Legal names for nn are X1,X2,...,X9,GO, or OV.

and the options are

FORMAT,value specifies the format of the file; U for unblocked, B for blocked, C for compressed. The default is unblocked for all files except GO; the default for GO is blocked.

FSIZE,value specifies the decimal length of the file in logical records. If ALL is input for a value, the remainder of the BT area will be allocated for this file. An ALL input is allowed only once, and is only allowed for Xi files (not GO or OV). A check is made for overflow of the BT area at the time the ALLOBT command is input. The default value is 1000 records. Note that the file size in sectors is computed using the logical record size and not the granule size.

RSIZE,value specifies the decimal number of words per logical record. This field is only meaningful for blocked or unblocked files, since the Monitor compresses records of compressed files into 256-word blocks. Blocked files have a default record size of 128 words, and unblocked files have a default record size equal to the granule size. Note that if RSIZE > 128, unblocked organization will always be given to the file.

GSIZE,value specifies the decimal number of words per granule. This field is only used in directly accessing a file. The default granule size will be the size of a RAD sector.

SAVE specifies that this file is to be saved throughout the job and not reallocated between job steps.

Example:

Change the default assignments of the background temp files:

The group of ALLOBT commands

```
5. !ALLOBT(FIL,OV),(FSIZ,Ø)
4. !ALLOB(FIL,X4),(FSI,ALL)
   !(GSIZE,180)
3. !ALLO(FILE,X3),(FSIZE,20);
   !(FSIZE,100),(RSIZE,30)
2. !ALL(FILE,X2),(FORMAT,B);
   !(FSIZE,1000),SAVE
1. !ALLOBT(FILE,X1),(FORMAT,C);
```

could be used by a background program to achieve the following results:

1. The X1 temp file would be a compressed file that could hold approximately 1000 EBCDIC cards. This file would be saved throughout the entire job.

2. The X2 temp file would be a blocked file which could hold a maximum of 100 binary cards.

3. The X3 temp file would be an unblocked file containing 40 sectors (assuming a 7204 RAD) with a granule size of 180 words or two sectors.

4. The X4 temp file would be an unblocked file with a record and granule size of 90 words (assuming a 7204 RAD) and would be allocated the remainder of the Background temp area.

5. The OV file would not be allocated.

After inputting this series of ALLOBT commands, the background temp area would have the following layout (assuming a 7204 RAD):

X2	X3	X4	X1	GO
36 Sectors	40 Sectors	n Sectors	120 Sectors	Default Size

Note that X4 receives n sectors, where n is the remainder of the area after all other files have been allocated. X1 is allocated at the opposite end of the BT area, since it will be saved throughout the entire job.

The formula used to calculate the number of RAD sectors for X2 is the following:

$$\frac{RSIZE}{256} \times FSIZE \times 3$$

where 256 is the number of words per blocking buffer and 3 is the number of RAD sectors (assuming a 7204 RAD) necessary to contain a blocking buffer.

The formula used to calculate the number of RAD sectors for X1 is

$$\frac{FSIZE}{25} \times 3$$

where it is assumed that 25 cards can be compressed into a 256-word blocking buffer. The number 3 is the number of RAD sectors necessary to contain a blocking buffer.

MODIFY The Monitor MODIFY control command is a special command used only at system boot time to modify (patch) one or more system modules. Its use must be preceded by sense switch settings 1 through 3. (See Appendix E "System Patching" for a complete description of control command format and use.)

DEBUG CONTROL COMMANDS

PMD The postmortem dump (PMD) control command causes the Monitor to dump a specified area of memory if a background job is aborted during execution. Such a dump is termed "postmortem" because it is performed after the background program has been aborted, terminated normally, or not executed at all for any reason. The dump is always output on the DO device. In the case of an abort the time to perform the dump is not included in the total time on the LIMIT control card. Note that the PMD command must precede the RUN command.

The form of the PMD command is

```
! PMD [U][,(from,to)][,(from,to)]...
```

where

- U specifies that an unconditional dump at the end of the job is to be output, even if there were no errors. If U is absent, the dump occurs only if the job is aborted.
- from specifies the location (in hexadecimal) at which dumping is to begin. If no locations are specified, the entire background is dumped.
- to specifies the last location (in hexadecimal) to be dumped. The last location must ≥ first location.

A maximum of four location pairs is processed and only the last PMD command is honored within a job step. If an error occurs anywhere on the command, the entire command must be reinput.

Example:

Request a postmortem dump:

```
! PMD U,(1200,1300),(2000,3000)
```

This example requests an unconditional dump at the termination of the next program to run in the background. Locations 1200_{16} through 1300_{16}, and 2000_{16} through 3000_{16} will be output on the DO device.

INPUT CONTROL COMMANDS

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

EOD The user may define blocks in a data deck by inserting EOD control commands at the end of each block. When an EOD command is encountered, the Monitor returns an EOD status. Any number of EOD commands may be used in a job and for any reason.

The form of the EOD command is

```
!EOD
```

FIN The FIN control command is used to specify the end of a stack of jobs. When the FIN command is encountered, the Monitor writes it on the listing log to inform the operator that all current jobs have been completed, types "BEGIN IDLE" on OC, and then enters the idle state. All time preceding the FIN command is charged to the previous job, if job accounting is being performed. All time from the FIN command to the next JOB command is charged to the idle account.

The form of the FIN command is

```
!FIN
```

UTILITY CONTROL COMMANDS

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

PFIL, PREC The file and record positioning commands are used to position a device within its current file. The PFIL command, which is only valid for magnetic tapes or RAD files, will leave the device positioned before the file mark in the appropriate direction. Only background devices (not dedicated to the foreground or IOEX) can be positioned.

The forms for the PFIL and PREC control commands are

```
{! PFIL }
{! PREC} [area,] name[,BACK][,n]
```

where

[area,] name specifies a system device name, operational label, or RAD area and file name of the device that is to be positioned. This must be the first item in the specification field.

BACK specifies that the direction of the positioning is backward. The default is forward.

n specifies the number of records to skip. The "n" parameter applies only to the PREC command and not to PFIL. The default is skip one record. The PFIL command always refers to one file.

Examples:

1. Position a RAD file to the end of the data:

```
! PFIL GO
```

This example could be used to position the GO file so additional object modules could be added to those already existing.

2. Position a RAD file:

```
! PREC D1,ABCD,30
```

This example would position RAD file ABCD 30 records forward from its current position.

SFIL The skip file command is used to skip one or more files on a magnetic tape unit, but cannot be used to position a RAD file. When skipping forward, the SFIL command leaves the device positioned following the specified EOF. When skipping backward, the device is positioned at the first record of the designated file.

The form of the SFIL control command is

```
! SFIL name[,BACK][,n]
```

where

name specifies a system device name or operational label of the device that is to be positioned. This must be the first item in the specification field.

BACK specifies that the direction of the positioning is backward. The default is forward.

n specifies the number of files to skip. The default is one file.

Example:

Skip tape files:

```
! SFIL 9TA82,BACK,4
```

This example would cause back skipping of four files on the designated 9-track magnetic tape.

REWIND The REWIND command is used to rewind a magnetic tape or a RAD file. It has no effect on other devices.

The form of the REWIND command is

```
! REWIND [area,] name
```

where

[area,] name specifies a system device name, operational label, or RAD area and file name of the device that is to be rewound.

Example:

Rewinding a tape

```
! REWIND 7TAE0
```

This example would rewind the designated 7-track tape.

UNLOAD The rewind manual (UNLOAD) command causes the specified magnetic tape to be rewound in manual mode. Operator intervention will be required to use the device again (i.e., depressing the ATTENTION and START switches on a tape drive). An UNLOAD command for a RAD file produces the same results as a REWIND command.

The form of the UNLOAD command is

```
!UNLOAD[area,] name
```

where

[area,] name specifies a system device name, operational label, or RAD area and file name of the device that is to be rewound in manual mode.

Example:

Unload a magnetic tape:

```
! UNLOAD 9TA83
```

This example would cause the designated 9-track tape to be rewound in manual mode.

WEOF The write end-of-file (WEOF) command causes an end-of-file mark to be written on the output device if an EOF is appropriate for the device. For magnetic tape, a tape mark is written; for a RAD file, a logical file mark is written, for paper tape, an EOD is written. The WEOF command is ignored for all other devices.

The form of the WEOF command is

```
! WEOF [area,] name[,n]
```

where

[area,] name specifies a system device name, operational label, or RAD area and file name of the device that is to receive the EOF.

n specifies the number of end-of-files to write. The default is one.

Examples:

1. Write end-of-file on magnetic tape:

```
! WEOF 9TA81,2
```

This example would write two EOFs on the designated 9-track magnetic tape.

2. Write end-of-file on RAD:

```
! WEOF GO
```

This example would write a logical EOF on the GO file at its current position. This would result in truncation of a file if the file was positioned at some point other than its end.

DAL The Dump Accounting Log command causes the contents of the Accounting Log to be printed on the LO device. The Accounting Log is kept on the AL file on the D1 area of the RAD. An option exists to purge the file after the dump is completed.

The form of the DAL command is

```
! DAL[PAL]
```

where

[PAL] specifies that the Accounting Log is to be purged after the dump is completed.

PROCESSOR CONTROL COMMANDS

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

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

User programs on the FP or BP areas of the RAD are called by

```
! RUN area,file name
```

where file name is the name of the program to be executed.

All system processors and user processors on the System Programs area of the RAD can be called for execution by the control command:

```
! name parameters
```

where

name is the RAD file name of the processor to be executed (e.g., FORTRANH, SYMBOL, SL-1, or MACRSYM). Note that the RAD file name for Macro-Symbol should be 'MACRSYM" since the JCP does special allocation of the BT area if the name MACRSYM is encountered.

parameters are optional parameters interpreted by each processor. Normally, at least one input option and one output option must be specified. The options for all system processors recognized by RBM are defined in Table 6.

Example:

```
! MACRSYM SI,LO,CI,BO
```

This example specifies that control is to be given to the Macro-Symbol assembler. It also specifies that symbolic

Table 6. Processor Specification Options

Specification	Use	Used by
BA	Selects batch assembly mode.	Macro-Symbol
BO	Relocatable binary output on the BO device.	FORTRAN IV-H, SL-1, Symbol, Macro-Symbol
CI	Compressed input from the CI device.	Macro-Symbol
CN	Concordance listing.	Symbol
CO	Compressed output on the CO device.	Macro-Symbol
D	Debug mode compilation.	FORTRAN IV-H
GO	Relocatable binary output to temporary RAD storage (i.e., the GO file).	Symbol, Macro-Symbol, SL-1, FORTRAN IV-H
LO	Listing output produced on the LO device.	FORTRAN IV-H, Symbol, Macro-Symbol, SL-1
LS	Source listing produced on the LO device	FORTRAN IV-H, SL-1
LU	Listing of the update decks (if any) produced on the LO device.	Macro-Symbol
S	S in column 1	FORTRAN IV-H, SL-1
SI	Symbolic input from the SI device.	Macro-Symbol, FORTRAN IV-H, SL-1
SO	Symbolic (source) output produced on the SO device.	SL-1

input is to be taken 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; compressed input is to be received from the device to which the CI operational label is assigned; and binary output is to be transmitted to the device to which the BO operational label is assigned.

Upon reading this control command, the JCP will set up the blocking buffers and RAD Background Temp files for the processor, load the processor's root, and transfer control to the entry address of the root.

PROCESSOR INTERFACE WITH RBM

The system processors under RBM are

Macro-Symbol

Overlay Loader

RAD Editor

FORTRAN IV-H

Symbol

SL-1

System processors and any user processors on the System Programs area of the RAD should follow these common ground rules.

1. All processors must reside on the System Programs area of the RAD to be callable by an ! Name command. A user wishing to test a new version of a processor without destroying the permanent version could execute the processor from the OV file via an ! ROV command.

2. All processors must operate in the background space.

3. All system DCBs (M:DCBs) should be identified as a primary reference in the processor, since at load time, the Overlay Loader will furnish the processor with a copy of the system DCBs.

4. All processors with overlay segments need only make the explicit call to SEGLOAD to load the segments. The DCB used to load segment M:SL will be furnished by the Overlay Loader.

5. RBM will furnish the start address and end address of unused background memory to any processor that needs this information. The two addresses will be in the following locations, and should be defined via the EQU directive in the processor:

Location	Mnemonic	Description
X'153'[†]	K:BPEND	LWA + 1[††] of the background program's loaded area; that is, this cell contains the FWA[††] the processor can use for a dynamic table area.
X'141'[†]	K:BCKEND	LWA of usable background memory for the processor; that is, this cell contains the LWA the processor can use for a dynamic table area.

[†] All these addresses are in bits 15-31 with bits 0-14 containing zeros.

[††] LWA and FWA are the last word address and first word address.

6. If a processor has parameters to process from the "! Name" control command (where "name" is the processor's name) the address of the buffer containing the control command is in cell X'144'. That is,

Location	Mnemonic	Description
X'144'†	K:CCBUF	Address of control card buffer.

7. A processor must perform its own vertical format control of the printer if format control is required. That is, the processor must set the VFC (vertical format control) bit in the DCB via the Monitor Device Format Control call and ensure that the first byte output to the printer is a format control byte. If a processor (i.e., Macro-Symbol) outputs a title at the top of each page, the number of lines to print per page is contained in the following system call:

Location	Mnemonic	Description
X'174' byte 0	K:PAGE	Number of lines per page to print.

8. If a processor uses scratch files (Background Temp files X1-X9) and desires a different record size, granule size, or organization than is given by default by the JCP, the processor must make the appropriate system call on the Device Mode function. By calling the Device Mode function, the processor can set the file organization (blocked or compressed) and the appropriate record size and granule size. The Background Temp file default assignments by the JCP are described below.

9. In general, the processor should terminate input from SI when an end-of-file status is sensed on SI. To terminate, the processor should make a system call on EXIT. EXIT will close all the processor's DCBs and close all open RAD files.

10. All processors using the GO file should open GO and then do a file skip (PFIL function call) on GO so the GO file is properly positioned to receive additional data. The Job Control Processor will purge the GO file upon reading a JOB control command.

The Job Control Processor will automatically allocate one blocking buffer for each system DCB (M:xx) assigned to a blocked file. Additionally, the JCP will always allocate a blocking buffer if the GO file is used, and allocates a maximum of two blocking buffers for all Xi ($1 \leq 1 \leq 9$) files used. If a system or user process or is not satisfied with the blocking buffer allocation, a POOL control command can be used to override the default allocation.

The JCP will also allocate the Background Temp area of the RAD for all Xi files, where $1 \leq i \leq 9$. The GO and OV files will receive their SYSGEN defined sizes, unless overridden with an ALLOBT command. The GO file will be defined as a blocked file with a logical record size of 120 bytes; the OV file will be unblocked with the record and granule sizes equal to the RAD sector size. The JCP will scan all system DCBs and determine which of the Xi files are used. The Background Temp area that remains after GO and OV have been allocated will then be equally distributed among these Xi files. All Xi files will be given unblocked organization with the record and granule sizes equal to the RAD sector size. The user can override any of these defaults via an ALLOBT command. If the user desires not to have the Xi files reallocated between processors, the SAVE option on the ALLOBT command can be used.

†All these addresses are in bits 15-31 with bits 0-14 containing zeros.

3. OPERATOR COMMUNICATION

When events take place in the system requiring operator intervention, or when one job completes and another job begins, RBM informs the operator of these conditions by messages output to the operator's console (OC device). All such messages from the Monitor begin with two exclamation marks (!!). Generally, these messages require no operator response on the typewriter, but may indicate that some peripheral needs attention.

RBM MESSAGES

The messages itemized in Table 7 are output by the Monitor on the OC device.

Table 7. Monitor Messages

Message	Meaning
!!BACKG CKPT	Background has been checkpointed as a result of a foreground program load.
!!BCKG RESTART	Background has been restarted from its point of interruption.
!!BACKG USED BY FGD	Background space is being used by the foreground but a checkpoint was not required, since the background was inactive at the time of the foreground load.
!!BEGIN WAIT	Background has executed a "WAIT" request. An unsolicited key-in of "C" will continue background processing.
!!CK AREA TOO SMALL	An attempt was made to checkpoint the background, but not enough space was available on the CK area of the RAD. The background space will nevertheless be released to the foreground and the active background job will be aborted when the background is restarted.
!!CKPT WAITING FOR BCKG I/O RUNDOWN	The checkpoint function is waiting for all background I/O to run down so that the checkpoint of background can be completed.
!!CORE USED, CAN'T LOAD xxxxxxxx	The specified foreground program cannot be loaded for execution because the core space required for its execution is already in use.
!!FGD AREA ACTIVE	An FMEM key-in request cannot be honored because a foreground program is still active in the area being released.
!!FGT FULL, CAN'T LOAD xxxxxxxx	The specified foreground program cannot be loaded for execution because no room exists in the Foreground Programs Table.
!!FILE NAME ERR	A problem has occurred from a STDLB key-in request in attempting to open or close a RAD file.
!!I/O ERR, CAN'T LOAD xxxxxxxx	An I/O error occurred in attempting to load the specified foreground program for execution.
!!I/O ERR ON CKPT	An attempt was made to checkpoint the background, but a RAD I/O error occurred during the process. The background space will nevertheless be released to the foreground, and the active background job will be aborted when the background is restarted.
!!JOB ABORTED AT yyyyy	Background job has been aborted. The "yyyyy" parameter contains the address of the last instruction executed in the background. If aborted because the specified limit on a !LIMIT control command has been reached, the yyyyy parameter will contain the word "LIMIT".
!!KEY ERROR	Monitor cannot recognize an unsolicited key-in response. A new key-in should be attempted.
!!LOADED PROG xxxxxxxx	The specified foreground programs have been loaded for execution by the foreground loader. A maximum of three program names will be output in the one message.
!!NONEXIST., CAN'T LOAD xxxxxxxx	The specified foreground program cannot be loaded for execution because it does not exist on the RAD, or a Public Library required by the program

Table 7. Monitor Messages (cont.)

Message	Meaning
	does not exist on RAD. The foreground program must exist in the FP area or the OV file.
!!NOT ENUF BCKG SPACE	Insufficient background space to load the requested background program.
!!PAUSE comments	A !PAUSE control command card has been read. The comments field may contain tape mounting instructions. A key-in of "C" after pressing the INTERRUPT switch will cause RBM to continue reading from the job stack.
!!PROG xxxxxxxx RELEASED	The specified foreground program has been released.
!!PUB LIB, CAN'T LOAD xxxxxxxx	The request to load the specified Public Library for execution is not valid, since all Public Libraries must be automatically loaded by the system, as needed.
!!RLS NAME NA	A key-in request has been made to release a foreground program but the name of the program is not recognized by the system.
!!SIGMA 5/7 RBM-2, VERSION xxxx	This message is output on the OC device every time the system is booted from the RAD. The message can be terminated prematurely by hitting the BREAK key on the typewriter.
!!UNABLE TO CLOSE DCB xxxxxxxx	The specified DCB was not closed upon releasing a foreground program.
!!UNABLE TO DO ASSIGN	An !ASSIGN command cannot be fulfilled because either the DCB cannot be found, or the DCB is only five words in length, and a seven-word DCB is required (seven-word DCBs are required for any RAD file assign).
!!UNABLE TO LOAD BCKG PUB LIB	The current attempt to execute a background program has failed because the Public Libraries required by the background program could not be loaded. The current background job is aborted.
!!UNABLE TO TRIGGER CONTROL TASK INT.	This alarm is output to OC after the system is booted from the RAD if the RBM Control Task interrupt cannot be triggered.
!!yyndd ERROR	A parity or transmission error has occurred on this device. Any automatic retries that were specified have been performed before this message was output.
!!yyndd MANUAL	Device specified is in manual mode and may be out of paper, cards, or tape.
!!yyndd UNRECOG	Some condition on device type yy with physical device number ndd (hexadecimal) has caused the device to become not operational.
!!yyndd WRT PROT	Indicated unit is write-protected. If a magnetic tape, insert the write ring and make the appropriate key-in to retry the operation. If a RAD is specified, an SY key-in is required before the RAD can be written on.

TRAP HANDLER MESSAGES

The following messages are output by the trap handler upon occurrence of the various traps if the user does not specify his own trap handling:

!!MEM. PROT. ERR AT xxxxx

!!PRIVILEGE INST. AT xxxxx

!!NONEXIST. ADD. AT xxxxx

!!NONEXIST. INST. AT xxxxx

!!UNIMPLE. INST. AT xxxxx

!!STACK OVERFLOW AT xxxxx

!!ARITH. FAULT AT xxxxx

!!WDOG TIMER RUNOUT AT xxxxx

!!ILL. PARAM., CAL AT xxxxx

Note that the !ARITH. FAULT AT xxxxx" message is output for the fixed point arithmetic overflow trap, the floating-point fault trap, and the decimal arithmetic fault trap. The "!!ILL. PARAM., CAL AT xxxxx" message is output if a user program furnishes the Monitor an invalid parameter while attempting to use a Monitor function.

JCP MESSAGES

The messages itemized in Table 8 are output by the Job Control Processor on both the OC and LL devices.

Table 8. JCP Messages

Message	Meaning
!!BEGIN IDLE	Job Control Processor has read a !FIN card, which completes a job stack. The background then goes into an idle state. Processing will resume on a new job stack following an unsolicited key-in of C.
!!BI CKSM ERR !!BI SEQ ERR	JCP Loader encountered a checksum or sequence error on a binary card during the loading process.
!!BT OVERFLOW	Insufficient Background Temp RAD space to execute the requested background program. The job is aborted.
!!CC ERROR, BT OVERFLOW	The file size input on an !ALLOBT command is greater than the available Background Temp RAD space.
!!CC ERROR, FG KEY-IN REQUIRED	A request has been made to run a foreground program without previously inputting an FG key-in. The !RUN or !ROV command must be reentered after the FG key-in is input.
!!CC ERROR, ILL. RELOCATION OF BT	An improper !ALLOBT command was input to change a Background Temp (BT) scratch file that was designated as a "saved" file prior to this job step.
!!CC ERROR IN ITEM xx	An error exists in a JCP control command in the indicated item. Every item (except the ! character) followed by a blank or comma is counted in determining the item in error.
!!EOT ON FILE xxxxxxxx	End-of-Tape status was returned from an attempt to read or write the indicated RAD file.
!!ERR, CONTROL BYTE = xx	JCP Loader is not equipped to process the indicated control byte.
!!FGT FULL, CAN'T LOAD xxxxxxxx	The indicated foreground program cannot be loaded because insufficient space exists in the Foreground Program Table.
!!FILE xxxxxxxx NONEXIST.	The indicated RAD file was never allocated via the RAD Editor or was never written into.
!!ILL. DEFINE FIELD ITEM	JCP Loader has encountered a define field item that it is not equipped to handle.
!!ILLEGAL BINARY CARD	An EBCDIC card was read by the JCP Loader where a binary card was expected.
!!ILL. EXPRESSION	JCP Loader has encountered an expression that it is not equipped to evaluate (a mixed resolution expression). The load will be aborted.
!!ILL. NEG. ORG ITEM	JCP Loader has encountered an origin item that it is not equipped to handle (an origin item that moves the load location counter in a negative direction). The load will be aborted.
!!JCP	The JCP has just begun to read control commands. This occurs both at the beginning of a job and between steps within a job. If C is assigned to the typewriter or if "TY" override is in effect, the input light on the typewriter will indicate that RBM is ready for input of a control command. This message is output only to OC.
!!NOT ENUF SPACE FOR LOAD	JCP Loader is unable to complete the load because of insufficient background space.
!!PUB LIB, CAN'T LOAD xxxxxxxx	The designated program on the !RUN command is a Public Library and cannot be executed via a !RUN command.
!!SCHING FOR JOB CMD	The present job has been aborted and the JCP is searching the job stack for the next !JOB or !FIN command.
!!TOO MANY CONTROL SECT.	JCP Loader has encountered more than one nonstandard control section. The load will be aborted.
!!TOO MANY DCB'S	The maximum number of M: and F: DCBs was exceeded during the loading process. Approximately 27 DCBs can be accommodated by the system. The excess DCBs will not be stored in the DCB table or the RAD file header.

Table 8. JCP Messages (cont.)

Message	Meaning
!!TOO MANY DEF/REF'S	JCP Loader has encountered more than 255 declarations in the object module being loaded.
!!UNSATISFIED REF xxxxxxxx	Indicated REF was not satisfied during the loading process. This alarm occurs only on LL if no map was requested, or on LO if a map was requested.
!!UNSATISFIED REF'S DURING LOAD	This message is typed to the operator on OC at the end of a load if any unsatisfied REFs were encountered during the loading process.

UNSOLICITED KEY-INS

Unsolicited key-ins provide the operator a means of controlling a background job or of loading for execution or releasing foreground programs. Note that any control the operator can exercise over the foreground is provided through operator key-ins, so that foreground control is independent of the background job stack.

The operator can initiate an unsolicited key-in at any time by depressing the INTERRUPT switch on the control panel console. This action causes the Control Panel Task to be activated; the Control Panel Task, in turn, triggers the RBM Control Task. When the RBM Control Task becomes the highest priority task in the system (that is, when all foreground tasks are inactive), the message

!!KEY-IN

will occur on the OC device.

All operator responses will terminate with a NEW LINE code. A blank (i.e., space) is used as a field delimiter, and any number of blanks can be used to separate fields. A message can be deleted prior to the NEW LINE input by depressing the End-of-Message key.

The analysis and subsequent action from an unsolicited key-in is performed at the RBM Control Task priority level. If the operator response is not recognized as a valid input, the message

!!KEY ERR

is output on OC. In this case, the operator should retype the response. Note that if the typewriter is busy at the time of the Control Panel Interrupt (i.e., waiting for an input to complete), the operator must complete the input before the !!KEY-IN type-out can occur.

The specific responses to the !!KEY-IN type-out are listed in the following sections. All key-ins can be preceded by an optional exclamation mark.

C The Continue key-in directs the Monitor to either start processing the background job stack or to continue processing in the background. The C key-in will end an idle state or will continue a job after the job was discontinued by a wait key-in, wait system call, or PAUSE control command.

The C key-in has the form

C ⓝⓛ

COC The Continue from OC key-in is used to correct an errored control command from the OC device. If a processor reads an incorrect control command during an attended run, the WAIT state will be entered. For some processors such as the Overlay Loader and RAD Editor, the control command in error can be reinput from OC via the COC key-in after interrupting out of the WAIT state. After correcting the command in error, control will be transferred back to the C device.

The COC key-in has the form

COC ⓝⓛ

W The Wait key-in causes the current background job to be discontinued and enter a WAIT state.

The W key-in has the form

W ⓝⓛ

X The X key-in will abort the background job with any dumps that were requested. A message will be printed on OC and LL which shows the last background location that was executed.

The X key-in has the form

X ⓝⓛ

SY The SY key-in allows any RAD file to be written into by a background program by providing the operator a means of overriding the normal software protection of RAD files. The SY key-in is cleared by a JOB command.

The SY key-in has the form

SY ⓝⓛ

TY The TY key-in causes the C operational label to be assigned to the OC device. The next and all ensuing control commands will be read from OC. Control will be returned to the C device on a CC control command or key-in.

The TY key-in has the form

TY ⓝⓛ

CC The CC key-in is used in conjunction with the TY key-in and transfers control back to the C device by reassigning the C op label to its previous assignment.

The CC key-in has the form

CC ⓝⓛ

DT The DT key-in is used to inform the Monitor of the current date and time.

The DT key-in has the form

DT month, day, year, hour, minute ⓝⓛ

where

month specifies the current month ($1 \leq month \leq 12$).

day specifies the current day ($1 \leq day \leq 31$).

year specifies the current year ($00 \leq year \leq 99$).

hour specifies the current hour ($0 \leq hour \leq 23$).

minute specifies the current minute ($0 \leq minute \leq 59$).

RUN The RUN key-in is used to load and initiate a foreground program. The program will be loaded at the priority of the Control Task if the space required for execution of the program is available. If the space is not available, an alarm will be typed and the operator will have to retype the RUN key-in when the space is available.

The form of the RUN key-in is

RUN name ⓝⓛ

where name is the file name of the foreground program to be loaded. The program must exist in core image format in the Foreground Programs area of the RAD.

A !!KEY ERR type-out will occur from this key-in if the requested program is already loaded or if space was not available in the Foreground Program Table.

RLS The RLS key-in is used to release a foreground program. The interrupts associated with the program are disarmed and after I/O has run down for the program, the memory space occupied by the program is marked as not used.

The RLS key-in has the form

RLS name ⓝⓛ

where name is the file name of the foreground program to be released.

STDLB The STDLB key-in is used to change the permanent assignment of an operational label. For a "C" operational label, both the permanent and temporary assignments are changed, unless the "C" label is assigned to zero. In this case, only the temporary label is changed. The assignment will stay in effect until the system is rebooted from the RAD.

The STDLB key-in has the form

STDLB label [, area], name ⓝⓛ

where

label specifies one of the standard operational labels that was input during SYSGEN.

area specifies a RAD area for the case of assignment of an operational label to a RAD file. If the operational label is being assigned to a file in the BT area, the "area" input is optional.

name specifies a physical device name to which the operational label is to be permanently assigned, a numeric zero, the name of a RAD file, or another operational label. In the latter case, the first operational label will receive the same assignment as the permanent assignment of the second operational label. If "0" is specified, there is no permanent assignment, which means that all output is suppressed to that label.

INTLB The INTLB key-in is used to change the assignment of interrupt labels specified at SYSGEN time.

The INTLB key-in has the form

INTLB label, loc ⓝⓛ

where

label specifies one of the interrupt labels defined at SYSGEN time.

loc specifies the new absolute hexadecimal location to be associated with the label. The location n must be $58_{16} \leq n \leq y$, where y is the highest interrupt location specified at SYSGEN.

CINT The CINT key-in is used to disarm, arm and enable, or trigger a specified interrupt.

The CINT key-in has the form

$$\text{CINT } \begin{Bmatrix} \text{location} \\ \text{label} \end{Bmatrix}, \begin{Bmatrix} D \\ A \\ T \end{Bmatrix} \text{ ⓝⓛ}$$

where

location specifies the hexadecimal address of the interrupt to be modified. The location n must be $58_{16} \leq n \leq y$, where y is the highest interrupt location specified at SYSGEN.

label specifies an interrupt label.

D disarm the specified interrupt.

A arm and enable the specified interrupt.

T arm and enable, and trigger the specified interrupt.

Note that the location being acted upon must contain an XPSD instruction or the request will be rejected.

FMEM The FMEM key-in is used to change the boundary between background and foreground memory. The key-in will not take effect until the current background job step completes.

The FMEM key-in has the form

FMEM [n] ⓃⓁ

where n specifies the number of pages to be allocated to foreground memory. If n is less than the present foreground allocation, no active foreground programs can exist in the area being released. If the foreground area is not free, an alarm (!!FGD AREA ACTIVE) will be typed. If n is zero, the entire foreground area will be allocated to the background. The default case will be to allocate the number of pages specified during SYSGEN.

FG The FG key-in allows a foreground program to be loaded for execution from the background job stack via a RUN control command, and protects the foreground from inadvertently being destroyed by an error in the background job stack. The FG key-in must precede the RUN control command or the background job will be aborted.

The FG key-in has the form

FG ⓃⓁ

COMBINED KEY-INS To facilitate the key-in process, the following combinations of key-ins are allowed:

Combined Form	Result
FGC	Executes the FG and C key-ins
SYC	Executes the SY and C key-ins
SFC or FSC	Executes the FG, SY, and C key-ins
TYC	Executes the TY and C key-ins

DM, DB, DF The Dump Monitor (DM), Dump Background (DB), and Dump Foreground (DF) key-ins allow the user to dump the contents of core memory onto the device that is permanently assigned to the DO operational label.

These key-ins have the form

$\left\{\begin{matrix}DM\\DF\\DB\end{matrix}\right\}$ [from, to] ⓃⓁ

If the [from, to] field is absent, DM causes the entire Monitor area of memory to be dumped; DB causes the entire, currently defined background area to be dumped; and DF causes the entire, currently defined foreground area to be dumped. The presence of the [from, to] field indicates the first word address and last word address (in that order) of memory that is to be dumped. If the [from, to] field is present, any of the three key-ins (DM, DF, or DB) can be used to achieve the same result, since only the specified locations are dumped. The last word address must be equal to or greater than the first word address.

DED The DED key-in allows a device and, optionally, all other devices on the same IOP to be dedicated to the foreground or to IOEX.

The DED key-in has the form

DED yyndd $\left\{\begin{matrix}F\\X\end{matrix}\right\}$ [,I] ⓃⓁ

where

yyndd is the name of the device to be dedicated (e.g., CRA03, DCAF0, etc.).

$\left\{\begin{matrix}F\\X\end{matrix}\right\}$ specifies the device is to be dedicated to the foreground (F) or to IOEX (X).

I specifies all devices on IOP n (n of yyndd) are to be dedicated. If I is absent, only the one device for a single unit controller is dedicated, or all devices on the same multiunit controller are dedicated.

UND The UND key-in undedicates a device or IOP that was previously dedicated through a DED key-in. If the device was dedicated to IOEX, it should be undedicated from IOEX by using the X option.

The UND key-in has the form

UND yyndd $\left\{\begin{matrix}F\\X\end{matrix}\right\}$ [,I] ⓃⓁ

where

yyndd is the name of the device to be undedicated (e.g., CRA03, DCAF0, etc.).

$\left\{\begin{matrix}F\\X\end{matrix}\right\}$ specifies the device is to be undedicated from the foreground (F) or from IOEX (X). The same option that was used in dedicating the device should be used in undedicating the device.

I specifies that all devices on IOP n (n of yyndd) should be similarly undedicated.

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 "ASSIGN", Chapter 2).

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

ERROR (error was detected on operation)

or

MANUAL (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 conditions, 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 9).

㊓ is the NEW LINE code.

Table 9. Monitor Actions

a	Monitor Action
C	Continue "as is".
E	Inform the user program of the error and transmit record "as is".
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 8) depends on the circumstances.

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

!! CRndd ERROR

or

!!CRndd TIMED OUT

on the OC device, depending on where in the cycle the error took place. 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, if the hopper is empty, or if the device is in the manual mode, the Monitor outputs the message

!! CRndd MANUAL

on the OC device. The operator corrects the condition and then presses the START button on the card reader.

CARD PUNCH

Instead of outputting an error message when a punch error is first detected, the I/O handler attempts to punch a card x times (x = NRT, a DCB parameter specified by the user; see Chapter 4) 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. 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 MANUAL

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

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

!! CPndd ERROR

or

!! CPndd TIMED OUT

on the OC device, depending on where in the cycle the error took place. 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

When an irrecoverable 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 = NRT, a DCB variable specified by the I/O user; see Chapter 4) 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, if the carriage is inoperative, or if the device is in the manual mode, the Monitor outputs the message

!! LPndd MANUAL

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

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

!! LPndd UNRECOG

on the OC device. The operator should correct the condition and respond with the key-in

LPndd R (NL)

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 operator responds with an I/O key-in message. The action character selected depends on the circumstances.

PAPER TAPE PUNCH

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

!! PPndd MANUAL

on the OC device. The operator corrects the condition and depresses the START key.

If the paper tape punch is off-line or the power is off, the Monitor outputs the message

!! PPndd UNRECOG

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

PPndd C or R (NL)

MAGNETIC TAPE

If an error occurs during the reading or writing of magnetic tape, the Monitor I/O handler attempts a recovery x times (x = NRT, a DCB variable). If the error is irrecoverable, the user is informed via an error return.

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

!! MTndd UNRECOG

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

4. INPUT/OUTPUT OPERATIONS

The RBM I/O system provides the user with the capability of performing input/output operations on standard XDS peripheral devices. An I/O request is made through execution of a CAL1 instruction that addresses a Function Parameter Table (FPT), which in turn is a list of parameters that define the request. The FPT addresses a Data Control Block (DCB), which is a list of parameters that define the nature of the data file. The DCB then addresses a Device Control Table (DCT) entry or a RAD File Table (RFT) entry, depending upon whether the data file concerned is associated with a non-RAD peripheral device or a RAD file. The DCT entry contains the device status parameters, and the RFT entry contains the RAD file parameters.

The CAL1 instruction and FPT must be generated at assembly or compilation time. Symbol or Macro-Symbol users must include both the CAL1 and the FPT in the source code. For FORTRAN users, the compiler generates the necessary CAL1s and FPTs.

All DCBs are given names beginning with M: for system DCBs or F: for user DCBs. The DCBs may be included in the source code if desired. If not included, the Overlay Loader generates the DCBs necessary to satisfy any unsatisfied references to F: or M: DCB names. System DCBs generated by the Loader have default parameters; User DCBs generated by the Loader are left blank.

The correspondence between a DCB and either a device or RAD file can be established by using the ! ASSIGN control command. Other DCB parameters describing the data file may also be set by the ! ASSIGN control command.

Two types of Read/Write requests are provided. Type I requests have the completion status posted in the DCB. The disadvantage of this type of I/O operation is that a DCB cannot be shared among requests in different tasks because, in general, it is impossible to associate the completion status in the DCB with a specific request. For this reason, Type II requests are provided.

Type II requests result in the completion status being posted in the FPT associated with the request. This enables several requests (perhaps in several tasks) to be in progress simultaneously on a given DCB. Type II requests require that the associated FPT must be in memory and not in a register.

The CHECK function tests for the completion of READ/WRITE requests that are performed without waiting for completion. CHECK tests the completion status posted in the DCB (Type I requests), or FPT (Type II requests).

PERMANENT RAD FILES

Permanent files are defined through the RAD Editor by use of the :ALLOT command, and data can be entered through the RAD Editor or any program that uses the system I/O. At definition time, the following file parameters are given by the user:

File name (maximum 8 characters)

File organization (blocked, unblocked, compressed)

Record size (for blocked or unblocked files to be accessed sequentially)

Granule size (for files to be accessed directly)

File size

TEMPORARY RAD FILES

Temporary files are in the Background Temp area of the RAD and have the fixed names X_i ($1 \leq i \leq 9$), GO and OV. The size for these files can be set by using the ! ALLOBT control command. If no ! ALLOBT control command appears within a user job, the files assume default sizes that are set by the Job Control Processor. The files X_i should be considered as primarily for temporary use within a single job step, since they are all allocated from a single area with X_{i+1} beginning just above X_i. Therefore, changing the size of a file X_i can cause a change in location of files X_j for $j > i$. GO and OV are allocated from the top of the temporary file area downward. A change in the size of files X_i therefore has no effect on the RAD position of these files.

Since the size and location of temporary files can be changed through background job control commands, they must not be used by foreground programs.

FILE ORGANIZATION

BLOCKED FILES

Blocked files contain fixed length records whose length is less than or equal to 128 words. In blocked files, the largest possible integral number of records is combined into 256-word blocks. These blocks are basic units of data transmitted to and from the RAD. As sequential READ requests are made to a blocked RAD file, the blocks are read from the RAD into blocking buffers as necessary, and the data records are transmitted to the user's input buffer.

Blocked organization is specified for a file when the file is defined by the RAD Editor. A file specified by the user as blocked, but having a record size greater than 128 words, will be given unblocked organization.

UNBLOCKED FILES

Unblocked files contain records of fixed length, each of which begins on a sector boundary. Each record requires

some integral number of sectors that is the smallest possible integral number that can contain the record.

COMPRESSED FILES

Compression of EBCDIC data in RAD files is provided by RBM by the removal of blank characters, since many blanks occur in a typical programming language source code. Compressed files are blocked into 256 word blocks on the RAD and the records are of variable length. No record crosses a block boundary.

ACCESS METHODS

SEQUENTIAL ACCESS

The sequential access method provides record-by-record access to the file in the same way that a data file on magnetic tape is accessed. A sequential access READ/WRITE request results in the next record in sequence being read or written. Sequential access can be used on blocked, unblocked, or compressed files.

DIRECT ACCESS

In the direct access method, the user furnishes the relative granule number of the start of the READ/WRITE request and the number of bytes to be transferred. The user is responsible for the organization of the file, including discrimination of logical records, maintenance of a key structure within the file, etc. Addressing files by granules allows the direct access method to be independent of the RAD sector size. Granule size is specified by the user at file creation. Each granule begins on a sector boundary, and the RAD space between granule end and the beginning of the next sector is never involved in direct access to the file.

The user is not restricted to I/O operations whose length is less than or equal to the granule size. For requests of length greater than granule size, the I/O system chains I/O operations to effect a skip of the dead space.

I/O QUEUEING

The I/O system provides for queueing of all requests to I/O devices. That is, any I/O request (READ, WRITE, REW, etc.) requiring a device to be accessed results in the request for the specific access being queued.

Device requests are queued on a controller basis (one queue per controller), and they are queued in order by priority of the task making the request. For example, a READ request to a card reader will be placed in the queue for the specified card reader controller, and its position in the queue is determined by the priority of the requesting task and the relative priorities of the requests already in the queue. Requests for a designated device from a specified priority level are queued by order of occurrence. The queues are chains of entries representing requests for actual I/O operations on devices. There is a single pool of free entries for all devices, and these entries are removed from the pool and linked to the controller queues as needed. The queue entry is returned to the free entry pool when a queued request is completed.

At System Generation, the user may specify the maximum number of entries to be used for background requests to ensure that the background does not tie up all the queue entries, thus causing foreground requests to wait. Whenever a request is made and the free entry pool is empty (all queue entries in use), the request is made to wait until an entry is freed.

I/O CLEANUP AND I/O START

I/O Cleanup is the data processing performed between completion of the actual data transmission (signaled by occurrence of the I/O interrupt) and the completion of the request. It includes such functions as error testing, setup for error recovery, posting of completion status in the FPT or DCB, setting of indicators in the DCT, dequeueing the completed request, etc.

I/O Start is the operation of starting a device for the next request.

Under the RBM I/O system, CPU time is not taken from a task to perform data processing for lower priority tasks; instead, I/O Cleanup and I/O Start functions are performed at the various times and priority levels given below:

1. I/O Cleanup is performed at I/O interrupt time if either the current request or the highest priority request in the queue for the same device controller are from tasks of higher priority than that of the interrupted task. If I/O Cleanup is performed at this time, I/O Start will be performed if the highest priority request in the queue is from a task of higher priority than the interrupted task.

2. If a CHECKed request was not previously completed, the CHECK system instigates I/O Cleanup and I/O Start on the specified controller through to completion. The CHECK system performs at the priority of the CHECKing task.

3. At request time, I/O Cleanup and I/O Start are performed as necessary to satisfy the request at the priority level of the requesting task.

   For an I/O request with wait, the device is driven until the request is completed. I/O Cleanup and I/O Start are instigated as needed.

   For an I/O request with no wait (after queueing the request by priority), I/O Cleanup is performed if the device and controller are not busy. The device is thus made busy before control is returned to the user.

4. At the Control Task level, any I/O Cleanup or I/O Start that was deferred because of priority consideration at I/O interrupt time (item 1 above) is performed.

SHARING DCBs AMONG TASKS

DCBs can be shared among several tasks within a given program, subject to the restriction that no task can make a Type I request on a DCB that is busy with another Type I request.

DCBs explicitly referenced by the user are allocated and created by the user either in the source code (for Symbol and Macro-Symbol), the compiler, or the Overlay Loader. This means that each program has a private copy of all DCBs explicitly referenced, and no DCBs are shared among programs. The user program has the responsibility for coordinating the Open and Close functions for DCBs shared among tasks within a program. An Open request results in the DCB being opened if it is not already open. A Close request causes the DCB to be closed if it is not already closed. No attempt is made to balance Open and Close requests for a DCB to determine which Close request should actually cause the DCB to be closed.

SHARING I/O DEVICES AMONG TASKS

Any number of tasks within a given program can share any device by sharing a DCB assigned to the given device. For sequential type devices (i.e., card reader, card punch, line printer, magnetic tape, paper tape reader, paper tape punch), responsibility for positioning and/or determining the position of the device is left to the user. No attempt is made to analyze a request on a DCB to determine which task has made the request.

Sequential output devices (i.e., card punch, paper tape punch, line printer, magnetic tape), can be shared by tasks (possibly in different programs) that use different DCBs. If several DCBs are assigned to an input or output device such as a magnetic tape drive, only write requests may be made through these DCBs. The sharing of output devices by different programs using different DCBs is used for logging error conditions or alarms.

Sequential input devices (i.e., card reader, paper tape reader, magnetic tape) cannot have different DCBs assigned to them. Sharing of these devices must be accomplished through real-time requests on a single DCB. For example, a background user who wishes to use double buffering on a card reader can do so by using two real-time Read requests with two different FPTs.

Random access devices such as RADs can be shared, using direct access, by tasks within different programs using various DCBs. The sharing can be performed without restriction other than those restrictions normally imposed on tasks sharing a DCB.

As DCBs are opened and closed, a count of the DCBs that are open and assigned to a device is kept. This count is incremented for every open request on a DCB assigned to the particular device, and is decremented for each Close request.

SHARING RAD FILES AMONG TASKS

Any number of tasks within a program can share a RAD file by sharing a DCB assigned to the file (subject to the conditions placed on tasks sharing DCBs discussed previously). A RAD file shared in this manner can be accessed either sequentially or directly. Input/output Requests are allowed.

Tasks can share a RAD file using different DCBs with the restriction that no sequential input or blocked sequential output requests can be allowed on a file shared in this manner. A count of the number of DCBs opened and assigned to a RAD file is kept for each file. If the count is greater than one, no sequential input from or blocked sequential output to the file is allowed.

An Open request on a DCB assigned to a RAD file results in opening of the file if it is not already open. A Close request on such a DCB results in closing of the file only if the "Open DCB Count' for the file is 1.

I/O END ACTION

Foreground programs (for READ, WRITE, and IOEX requests) may use I/O end-action. Two types of end-action are possible:

1. The user provides an end-action address in the FPT. A transfer to this address will be made following the occurrence of an I/O interrupt that signals completion of the data transfer. This end-action transfer is made by executing

   ```
   BAL, 11 end-action address
   ```

   with the CPU in master mode, the I/O interrupt still active, and the AIO status in register 5. The end-action may destroy any registers except 5, 7, and 11. Return from the end-action routine must be made by

   ```
   B *11
   ```

   with the CPU still in master mode, the I/O interrupt still high, register 5 containing the AIO status, and register 7 containing the device type index (DCT).

   It should be noted that since end-action is performed with the I/O interrupt high, all tasks whose priority is lower than that of the I/O interrupt task are effectively disabled for the duration of the end-action.

   Since the end-action user can seriously degrade interrupt response for lower priority tasks, it is strongly recommended that this type of end-action not be used for applications where other techniques are satisfactory.

2. The user FPT contains either an interrupt number or interrupt label specifying a system interrupt. The system interrupt is triggered upon occurrence of an I/O interrupt that signals completion of the request (this interrupt will be triggered before the I/O interrupt is cleared). The task connected with the specified interrupt then performs the end-action function at the proper priority level. The user is responsible for connecting the interrupt and ensuring that it is armed and enabled.

   ~~If the end-action interrupt priority has the same priority as the requesting task when the interrupt is triggered~~, The I/O system sets a flag in the TCB to indicate that the trigger has been performed. The EXIT routine interrogates this flag before performing the EXIT for centrally connected tasks. If the flag is set, the occurence of the interrupt (previously lost by the triggering of an active interrupt) will be simulated. Directly connected tasks using this type of end-action assume the responsibility for solving problems of this type.

   No end-action is taken for requests that do not require actual device access (i.e., READ on a blocked RAD file that does not require reading a block from the RAD); however, end-action will be performed for requests that cause a device access and an I/O interrupt.

RESERVING I/O DEVICES FOR FOREGROUND USE

I/O devices can be reserved for exclusive use of the foreground program system through SYSGEN input, operator key-in, or through a system call from a foreground program. Reservation can be made either for a specific device or for all devices associated with a given IOP. A reservation for a device on a multidevice controller results in reservation of the entire controller. When a device is reserved, it is specified either that all foreground requests for the device will be allowed, or that only foreground direct access (IOEX) requests will be allowed.

Device reservation results in all background requests to the device being held in abeyance until the device is released for background use. The background user program is unaware that execution is suspended.

When devices are reserved for IOEX operation only, all regular (READ, WRITE, etc.) requests from the foreground are also held in abeyance. A foreground program making such a request will be pending in the I/O system until the device is released for regular foreground use. In IOEX reservation mode, the foreground task doing IOEX I/O and the task that eventually releases the device for regular foreground use must be of higher priority than any task doing standard I/O. This type of device reservation is designed for the user with a high throughput foreground application who needs to guarantee that a device (typically the RAD) is immediately available and is not in the process of an I/O operation for another task.

A count is kept of the number of reservations (STOPIO requests) of each type (either all foreground I/O allowed or IOEX only allowed) for each controller. As devices are reserved, the proper count is incremented, and as they are released, the count is decremented. A value greater than zero indicates that the controller is reserved. The user must balance each STOPIO request with a STARTIO request so the system can maintain order.

When I/O requests are received by the system, the reservation counters are tested to determine whether the request is allowed. Any request not allowed because of a STOPIO request will not be queued. Any request previously queued but currently not allowed will not be started.

The foreground user can specify in a STOPIO request that the in-process operation on the specified controller be aborted through execution of an HIO.

DIRECT I/O EXECUTION (IOEX)

RBM provides the foreground user with the capability of programming I/O devices by furnishing TIO, TDV, HIO and SIO instructions, and IOP command doublewords. The instructions are executed by the I/O system. Status information resulting from the instruction execution is returned to the calling task for analysis. No testing or error recovery is performed by the system.

Prior to performing IOEX operations on a device, the device must be reserved for IOEX operation only. This requires the user to issue a STOPIO unless it is known that the device was properly reserved either during SYSGEN or by the operator.

For SIO requests, if a user wishes to be informed of the occurrence of the I/O interrupt, it is imperative that the command doublewords be set up so that a channel end interrupt occurs.

The SIO request may also include an end-action address or an end-action interrupt number. This address is the entry to the user's routine that will analyze the resultant status and set appropriate indicators. It should be noted that the user's routine is entered with the I/O interrupt active, and to avoid seriously degrading system interrupt response, the routine should require as little execution time as possible. Control is transferred to the end-action routine through a

~~BAL, 11, address~~ BAL,11 address

If an end-action interrupt number or interrupt operational label is given, the interrupt will be triggered upon completion of the operation signaled by the I/O interrupt. This interrupt must be connected to the end-action task.

IOEX cannot be used by background programs. The requirement that background programs be able to read any data written by foreground programs is satisfied by the facility to treat an entire RAD area as a single-file for direct access input (see discussion of IOEX system call later in this chapter for details of the calling sequences and formats involved).

OPERATIONAL LABELS

Under RBM, operational labels are used to lend flexibility in the assignment of DCBs to peripheral devices. Operational labels represent logical devices and are assignable to physical devices and RAD files.

The system DCBs are defined in Table 10.

Table 10. System DCBs

DCB Name	Op Label or RAD File Assignment	Comments
M:C	C	The first 12 DCBs are assigned to the standard operational labels.
M:OC	OC	
M:LO	LO	
M:LL	LL	
M:DO	DO	
M:CO	CO	
M:BO	BO	
M:CI	CI	
M:SI	SI	
M:BI	BI	
M:SO	SO	
M:PL	PL	
M:Xi ($1 \le i \le 9$)	Xi	DCBs for Background Temp scratch file.
M:GO	GO	DCB to write on GO file.
M:OV	OV	Output DCB for Overlay Loader.
M:SL[t]	Appropriate program file	Input DCB for Segment Loader.

[t]The M:SL DCB does not have to be referenced by a program using overlays, since this DCB is automatically furnished by the Overlay Loader for any program with overlay segments.

DCBs can be assigned to operational labels, physical devices, or RAD files. Assignment of DCBs to operational labels is accomplished in one of the following ways:

1. The user can effect the assignment through the source code by allocating and defining the DCB, providing the operational label table index value (see "DCB Creation"), and specifying that the assignment is to an operational label.

2. The Overlay Loader provides each program with a copy of all system DCBs referenced by the program. These DCBs are distinguishable by the prefix "M:", and such DCBs are assigned to system operational labels (see the "Overlay Loader" chapter).

3. ! ASSIGN control commands can be used to assign DCBs to system operational labels (see the discussion of ! ASSIGN control command in Chapter 2).

Operational labels are assigned to physical devices or RAD files. Each operational label has two assignments, temporary and permanent. The permanent assignment is used by all foreground programs and serves as a default for the temporary assignment. The temporary assignment is used by all background programs. Assignment of operational labels to devices or RAD files is made in the following ways:

1. At System Generation, permanent assignments are made and remain in force until changed through STDLB key-in. The original permanent assignments are reinstated whenever the system is again booted and initialized.

2. The STDLB key-in can be used by the operator to change the permanent assignment of an operational label, which will result in a corresponding change in the temporary assignment when the next JOB card is read.

3. The !STDLB control command can be inserted by the user to change the temporary assignment of an operational label.

Temporary assignments remain in force only within a single background job. Each ! JOB control command causes the temporary assignments to be set the same as the permanent assignments except for the "C" operational label. Should temporary assignments be changed by an ! STDLB control command, the new assignment remains in force only for the duration of the job except for the "C" operational label.

At System Generation, the user may specify any number of optional operational labels, with the proviso that the optional labels be two characters in length. For each optional operational label, an entry is built in the operational label table, and each entry requires four bytes of system residence.

The relationship of system operational labels to their table index values is defined in the DCB description. The user assumes responsibility for determining the index values associated with any optional operational labels created at System Generation.

DCB CREATION

The Overlay Loader creates the DCBs for FORTRAN IV-H programs that reference the standard FORTRAN operational labels 105, 106, AND 108 for their I/O requests. For other labels, the user must create DCBs using ! ASSIGN control commands and machine language subroutines.

DCBs for Symbol and Macro-Symbol programs are allocated and defined in the following ways:

1. User Created DCBs: The user may create his DCBs in the source code. The parameters defined at source

time may be overridden by ! ASSIGN control commands if the user follows the convention of defining the name of a DCB and beginning the name with F:.

Warning: DCBs will not receive any memory protection, and assembly language users should exercise extreme care to prevent accidental alteration.

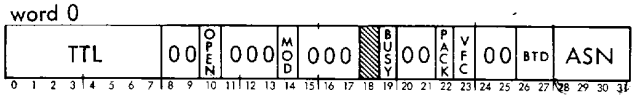
2. Loader Created DCBs: At the conclusion of the object module load and the library search and load, the Loader creates DCBs for any unsatisfied REFs beginning with M: or F:. For REFs to system DCBs (M:), a copy of the standard DCB is included in the root portion of the load module. This DCB contains standard system parameters, including standard assignment to a system operational label. For example, M:LO is assigned to operational label LO. User DCBs (F:) are included in the load module but they are left blank. The background user must define the parameters for F: DCBs through ! ASSIGN control commands. Definition and assignment of F: DCBs in foreground programs can be made through Overlay Loader control commands.

3. ! ASSIGN Command Created DCBs: ! ASSIGN control commands can create DCBs in addition to defining or redefining parameters in existing DCBs. This DCB creation facility enables FORTRAN IV-H programs to perform I/O using variables as operational labels. At run-time, the FORTRAN program evaluates such variables, converts the variable value to a DCB name and locates the DCB. For example, a FORTRAN variable with value 101 would result in an I/O operation using DCB F:101. The DCB must have been created in a Symbol or Macro-Symbol subroutine or through an !ASSIGN control command.

DCB FORMAT

The format for a Data Control Block is given below:

word 0



word 1

word 2

word 3

word 4

where

Word 0

TTL is the total length of the DCB. A length of seven words is required by any DCB which may be assigned to a RAD file. A length of five words is sufficient for DCBs that are assignable only to devices or operational labels.

TTL must be set by the mechanism creating the DCB; either by the user (through the source code), Overlay Loader (when the user declares the name but does not create the DCB), or by the ! ASSIGN control command which creates the DCB.

OPEN is the DCB open indicator. It must be set to zero before the DCB is opened. The I/O system sets the indicator to 1 when the DCB is opened.

MOD is the mode flag (0 for EBCDIC mode; 1 for binary). The flag is set at DCB creation time and its status may be changed by ! ASSIGN control commands or through a Device Mode system call. This flag has meaning only for I/O requests to 7-track magnetic tape, card punch, or card reader. For requests to read a card reader, Mode flag 0 causes a Read Automatic.[†] Input from a card reader designated as the C device is always performed in automatic mode (mode flag is ignored).

BUSY is the DCB busy indicator that is set and maintained by the I/O system to indicate that a Type I request using the DCB is in progress. Any Type I request using a DCB that is made when the DCB is busy will result in an error.

PACK is the indicator specifying packed binary format on 7-track magnetic tape when BIN mode is also specified (1 indicates packed; 0 indicates unpacked). PACK may be set using the ! ASSIGN control command. This indicator is also set to value 1 when a DCB is opened and that DCB is assigned to an operational label, which in turn is assigned to a 7-track magnetic tape.

VFC is the vertical format indicator (0 indicates no format control; 1 indicates format control) specifying whether or not the first character of an output record is to be used to control vertical positioning for output to a line printer or keyboard/printer. Under format control, the line printer is given a "print with format" order. The keyboard/printer performs a preliminary new line (regardless of the format character) and outputs the record beginning

[†] See Chapter 3, XDS Sigma Card Readers (Models 7120/7122/7140) Reference Manual, Publication No. 90 09 70.

with the second byte. On all other devices, the first byte is output as data. VFC has no effect on other I/O operations. This indicator is set at DCB creation by ! ASSIGN control command or through the Device Mode system call. The format control codes are itemized in Table 11.

BTD is the byte displacement specifying at which byte (0-3) in a buffer the data begins.

ASN is the assignment type indicator (0 means null; 1 means RAD file; 2 means not used; 3 means device or system operational label).

Word 1

NRT is the number of recovery tries to be allowed before outputting a device error message. This parameter is set at DCB creation or through an ! ASSIGN control command.

DEVF is an indicator specifying whether the device assignment (when ASN has value 3) in force is directly to a physical device or indirectly through an operational label (1 means direct; 0 means indirect) . This indicator may be set at DCB creation or by an ! ASSIGN control command. See TYPE and DEV/OPLB/RFILE discussion below.

L is an indicator specifying whether the assigned device is a line printer or keyboard/printer. The indicator is set by the system at OPEN time.

Table 11. 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.

TYPE is a field indicating the type of device that is directly assigned if ASN has value 3 and DEVF has value 1.

Value	Device
1	TY
2	PR
3	PP
4	CR
5	CP
6	LP
7	DC
8	9T
9	7T
10	CP (Low Cost)
11	LP (Low Cost)
13	PL

If ASN has value 1, TYPE specifies the area that contains the RAD file.

Value	Area
0	SP
1	FP
2	BP
3	BT
4	XA
6	D1
:	:
20	DF

DEV/OPLB/RFILE contains one of three:

1. The DCT index of the assigned device when the assignment is to a device (ASN equals 3 and DEVF equals 1),
2. The operational label table index of the assigned operational label when the assignment is to an operational label (ASN equals 3 and DEVF equals 0). The index values for standard system operational labels are

Label	Index Value
C	1
OC	2
LO	3
LL	4
DO	5
CO	6
BO	7
CI	8
SI	9
BI	10
SO	11
PL	12

The user is responsible for determining the index values for his optional operational labels. These

values are a function of the order in which the optional operational labels are specified as System Generation.

The index value for the devices are also a function of the order that the devices are specified at System Generation. The TYPE and DEV/OPLB/RFILE values are set at DCB creation or through ! ASSIGN control commands.

3. When a DCB is assigned to a RAD file (ASN equals 1), this field contains the index to the RFT (RAD File Table). This value is set when the DCB is opened. The RFT entry is created at OPEN if an entry does not already exist for the file.

Word 2

TYC is an indicator showing the type of completion for an I/O operation. TYC is set by the I/O system at the completion of each request that uses the DCB in a Type I mode (see discussion of Read and Write system calls below).

The completion type codes are

Code	Meaning
1	Normal
2	Lost Data
3	Beginning of Tape
5	End of Tape
6	End of Data
7	End of File
8	Read Error
9	Write Error
10	Write Protection Violation

BUF is the address of the user buffer for requests whose FPTs do not include a buffer address. The address is established at DCB creation by assembly language users who create their own DCBs. The parameter is not included in DCBs built by the Overlay Loader and is not set through ! ASSIGN control commands.

Word 3

RSZ is the default record size in bytes ($1 \leq RSZ \geq 32{,}767$). The parameter is used as the byte count for Read/Write requests that do not include a byte count. RSZ may be set at DCB creation, either through a Device Mode system call or through an ! ASSIGN control command.

ERA is the address of the user's routine that handles errors associated with insufficient or conflicting information in the DCB or FPT. Zeros in this field are used to indicate that no user error routine exists, (see discussion of error and abnormal returns below). This address can be established at DCB creation, but is more typically set through an OPEN system call.

Word 4

ARS is the actual record size in bytes. The parameter is set by the I/O system when a request is completed. It is set in the DCB for Type I requests only.

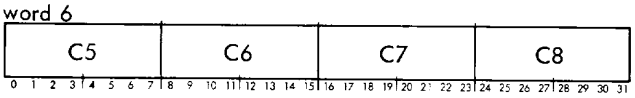
ABA is the address of the user's routine that handles abnormal conditions associated with insufficient or conflicting information in the DCB or FPT. Zeros are used to indicate that no user abnormal routine exists (see discussion of error and abnormal returns below). This address can be established at creation time or set through an OPEN system call.

If the TTL field in word 0 has a value of 5, word 4 terminates the DCB.

If the TTL field has a value of 7 or 90, and the DCB is assigned directly to a RAD file, words 5 and 6 have the following format:

word 5

word 6



where the C_i are the EBCDIC characters defining the RAD file name when the assignment is to a RAD file (ASN equals 1). The file name is left-justified and filled with trailing blanks. The name can be placed in the DCB at creation time or through ! ASSIGN control command. Note that DCBs assigned to RAD files at creation time, with the file names included, are not compatible with Batch Processing Monitor DCBs. Such DCBs can be used under BPM only if the assignment is reestablished through use of the ! ASSIGN control command. If the words 5 and 6 are all zero and ASN specifies assignment to a RAD file, the entire area specified in TYPE is taken as the file.

ERROR AND ABNORMAL CONDITIONS

Certain error codes are returned to the user's error or abnormal return routines upon occurrence of various conditions. At entry to these routines, the error code is contained in byte 0 of register 10, the DCB address is contained in the address field (low-order 17 bits) of register 10, and the address of the location following the CAL1 is contained in register 8.

Foreground users must provide error and abnormal returns on all I/O requests with wait and on all CHECK requests. If background users omit the error and abnormal addresses, the system will take action as detailed below. The error codes are defined in Table 12.

Table 12. Monitor Errors and Abnormal Returns

I/O Code (Hexadecimal)			
	D^{t} CB	F^{tt} PT	Meaning of I/O Codes
			Abnormal Conditions (Continue)
01	X		A DCB has been opened with incorrect parameters.
03	X		The assigned RAD file does not exist or the assigned device is down.
05		X	An end-of-data has been encountered (i.e., an EOD has been read).
06		X	An end-of-file has been encountered (i.e., a control command has been read on the C device).
07		X	The buffer specified is smaller than the data read.
0A	X		An attempt has been made to close a DCB that is already closed.
1C		X	The end-of-tape has been encountered.
1D		X	The beginning-of-tape has been encountered.
2E	X		An attempt has been made to open a DCB that is already open.
30		X	The request resulted in a condition which the operator can correct. The proper message has been output on OC.^{ttt}
			Error Conditions (Abort, with postmortem dump if specified)
40	X		A request has been made to read an output device.
41		X	An irrecoverable read error has occurred.
42		X	A RAD write protection violation has occurred.
44	X		A request has been made to write on an input device.
45		X	An irrecoverable write error has occurred.
46	X		The DCB contains insufficient information to open a closed DCB on a read operation.

Table 12. Monitor Error and Abnormal Returns (cont.)

I/O Code (Hexadecimal)			
	D^{t} CB	F^{tt} PT	Meaning of I/O Codes
			Error Conditions (Abort, with postmortem dump if specified)
47	X		The DCB contains insufficient information to open a closed DCB on a write operation.
48	X		A nonreal-time request was made on a busy DCB.
4A	X		The user buffer address is not valid or byte count is zero.
54	X		More than one attempt has been made to read a control message from the C device, through the same DCB.
55	X		The DCB cannot be opened because the RFT is full, the RAD is down, or no buffer could be found for the directory search.
58	X		A foreground request was made to the C device.
59	X		DCB has changed since being OPENed.
60	X		Input request on a shared device or file.

^{t}Returns due to insufficient information (error or abnormal address in DCB is honored).

^{tt}Returns due to device failure or abnormality (error or abnormal address in FPT is honored).

^{ttt}All background requests (even those specifying abnormal returns) will be held until the operator corrects the condition. This condition is never returned to the background.

I/O SYSTEM CALLS

OPEN A FILE

OPEN The OPEN system call opens the data file if it is not already open. If the addressed DCB is assigned to a device (directly or through an operational label), a count is kept of the number of open DCBs assigned to the device.

If the DCB is assigned to a RAD file, an entry is built in the RFT (RAD File Table) if one does not already exist, and the index of the entry is placed in the DEV/OPLB/RFILE field of the DCB. A count of open DCBs assigned to the RAD file is also maintained. The user may specify a buffer to be used in the RAD File directory search but this is not mandatory. If such a buffer is not given, the OPEN function will use available blocking buffers.

At OPEN, the error and abnormal addresses in the DCB may be set or changed. The OPEN function causes the specified DCB's file-open indicator (OPEN) to be set to 1.

If the specified DCB is assigned to an operational label and the operational label in turn, is assigned to a 7T device, PACK and BIN are set to 1.

If a DCB is already open (OPEN = 1) for device-assigned DCBs when the OPEN function is called, an abnormal condition is signaled (see Table 12). The device indicator (DEV/OPLB/RFILE) of the DCB is checked for validity. If it references a valid operational label or physical device, the DCB is marked open; if the device indicator is invalid, the DCB is not marked open and an abnormal condition is signaled (see below for the OPEN call format).

For DCBs assigned (directly or through op labels) to line printers or keyboard/printers, the L indicator in the DCB is set to 1.

CLOSE A FILE

CLOSE The CLOSE function closes a DCB by setting the DCB open indicator (OPEN) to 0, which may result in closing the assigned data file on a device or RAD file. the CLOSE function decrements the "open DCB count" in the proper DCT or RFT entry, and if the count becomes zero, the data file is closed.

If the data file is to be closed and is a RAD file opened for output, the directory entry for the RAD file is updated with the information from the RFT entry and the entry is deleted from the RFT table.

If the data file is to be closed and is a RAD file opened for input only, the entry is deleted from the RFT table.

Closing other types of data files requires no action.

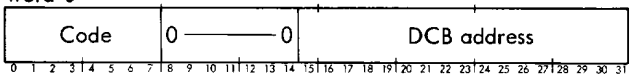
OPEN AND CLOSE SYSTEM CALL FORMAT

OPEN and CLOSE system calls have the format

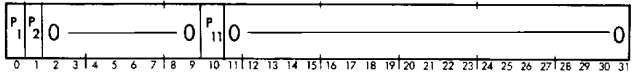
CAL1, 1 address

where address points to word 0 of the FPT shown below.

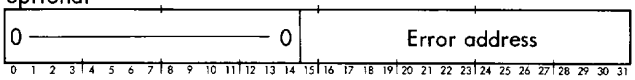
word 0



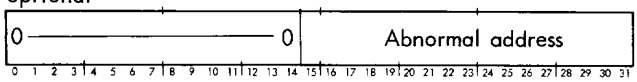
word 1



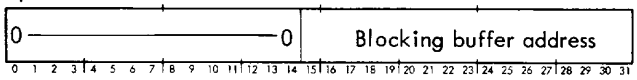
optional†



optional†



optional†



where

Word 0

Code is X'14' for OPEN, X'15' for CLOSE.

DCB address is the address of the associated DCB

Word 1

P_1 is the error address parameter presence indicator (0 means absent; 1 means present).

P_2 is the abnormal address parameter presence indicator (0 means absent; 1 means present).

P_{11} is the buffer address parameter presence indicator (0 means ~~present~~; 1 means ~~absent~~).

absent present

Word Options

Error address is the address of the entry to the user's routine that will handle error conditions.

Abnormal address is the address of the entry to the user's routine that will handle abnormal conditions.

Blocking Buffer address is the address of a 257-word buffer to be used for file directory search if the DCB is being opened to a RAD file.

CHECK I/O COMPLETION

CHECK The CHECK function tests the type of completion of an I/O operation initiated by a no-wait request. The user specifies addresses, which are entries to his routines, that handle error and abnormal conditions. At entry, register 10 contains the error or abnormal code as detailed in Table 12. Background users may take advantage of the standard system handling of the error and abnormal address in the FPT . The action taken by the system in this case is also detailed in Table 12. Foreground users must provide both error and abnormal addresses when checking (CHECK, no-wait) requests.

Users may specify a CHECK with no-wait by including a busy address in the FPT. This address is taken (with the address of the location following the CHECK CAL1 in

†In all FPTs for I/O functions where an optional parameter is not used, the parameter word must be omitted from the FPT and the corresponding presence indicator (P_n) set to 0.

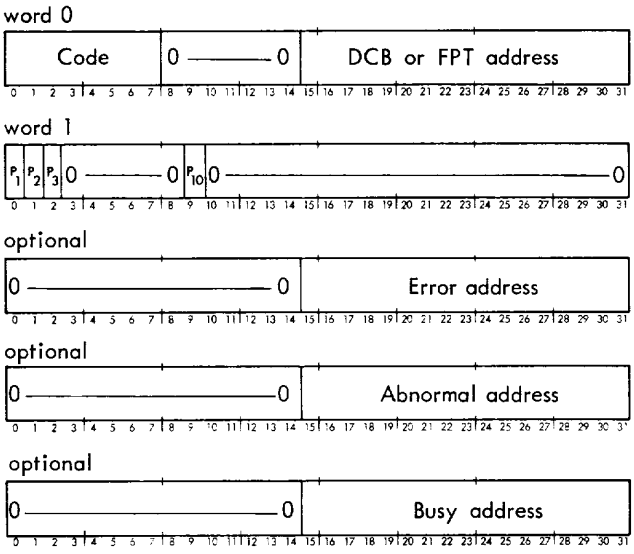
register 8), if the CHECKed operation is not complete. If no busy address is included in FPT, the CHECK function will wait for completion before taking the appropriate action.

The CHECK function (through its own FPT) addresses a DCB or an FPT, depending upon whether the request was Type I or Type II. The FPT associated with a request is addressed if the request was Type II and the completion parameters were posted in the FPT by the I/O system. A DCB is addressed if the request was Type I and the completion parameters were posted in the DCB.

The CHECK function call is of the form

CAL1, 1 address

where address points to word 0 of the FPT shown below.



where

Word 0

Code is X'29' for the CHECK function.

DCB or FPT address is the address of the DCB or FPT where the completion status is posted. P_{10} determines whether this field contains a DCB or FPT address.

Word 1

P_1 is the error address parameter presence indicator (1 means present; 0 means absent).

P_2 is the abnormal address parameter presence indicator (1 means present; 0 means absent).

P_3 is the busy address parameter presence indicator (1 means present; 0 means absent).

P_{10} is a code to determine whether a DCB or FPT is addressed (0 means DCB; 1 means FPT).

Word Options

Error address is the address of the entry to the user's routine that will handle error conditions.

Abnormal address is the address of the entry to the user's routine that will handle abnormal conditions.

Busy address is the address of the entry to the user's routine that will handle the "request busy" conditions.

READ A DATA RECORD

READ The READ function causes the I/O system to read a data record into a user buffer from the device or RAD file specified by the DCB.

If the addressed DCB is closed when the READ request is made, an implicit OPEN will be performed on the DCB.

READ requests may specify either a "wait for completion" or an "immediate return" condition. Foreground requests with wait must include error and abnormal returns in the FPT. Background requests can omit these addresses and have the system handle error and abnormal conditions. For requests with no-wait, such addresses in the READ FPT would be superfluous, since the user must perform a CHECK to test for error or abnormal conditions resulting from the request.

Should the input record be physically longer than the specified buffer length, data is lost and the user is notified through an abnormal return with code 07.

Should the input record be physically shorter than the specified buffer length, the buffer is not filled and the acutal record length is posted in the FPT or DCB.

Input from the card reader is performed either in automatic or binary mode. If the card reader is not the C device, the input mode is determined by the BIN flag in the DCB. The C device is always read in automatic mode. Foreground programs may not read the C device as this would disrupt the background job stream.

Input from the C device results in all control commands (! in column 1) being intercepted by the I/O system. Any control command other than ! EOD causes an abnormal return with a code of 06 in register 10. The input record is kept in the RBM control command buffer. If an attempt is made to read this same device again, an error return with code 54 is given (see Table 12).

An ! EOD record encountered from a card reader or paper tape reader on a READ request results in an abnormal return with code 05.

For random access input from RAD files, the user includes a key in the FPT. All READ requests without a key parameter are assumed to be sequential access requests and result in the next record in order being input into the user buffer. For sequential input from blocked files, a request without a key parameter may not result in an actual RAD access.

For sequential access input from compressed files, the I/O system decompresses the record in transmitting it to the user buffer.

Type II READ requests must include in their FPTs a completion status parameter in which the I/O system will post the type of completion code and the actual byte count.

A Type I READ request that finds the DCB busy with a previous Type I request results in an error condition (error code 48).

WRITE A DATA RECORD

WRITE The WRITE function causes the I/O system to write a data record from a user buffer to the device or RAD file specified by the DCB.

WRITE requests may specify "wait" or "no-wait". As with READ requests, WRITE requests specifying wait must include error and abnormal return addresses in the FPT. For requests with no-wait, such addresses in the FPT would be superfluous since the user must perform a CHECK to test for error and abnormal conditions resulting from the request.

For random access output to RAD files, the key address parameter is included in the FPT. All WRITE requests without a key address parameter present are assumed to be sequential access requests.

For output to compressed files, the I/O system compresses the record in transmitting it to the system blocking buffer.

Type II Write requests must include a completion status parameter word in their FPTs in which the I/O system will post the type of completion code and the actual byte count.

A Type I request that finds the DCB busy with a previous Type I request results in an error condition (error code 48).

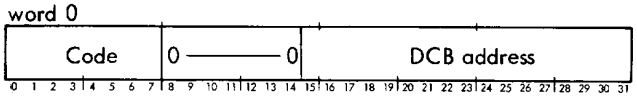
READ AND WRITE FUNCTION CALL FORMAT

Calls for these functions are of the form

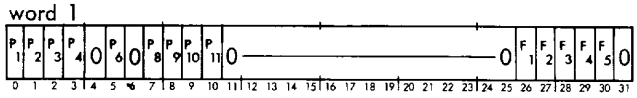
CAL1, 1 address

where address points to word 0 of the FPT shown below.

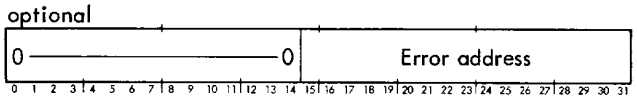
word 0



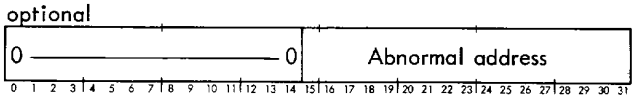
word 1



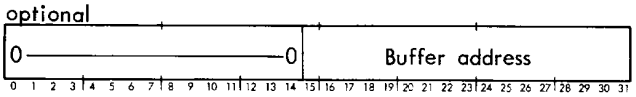
optional



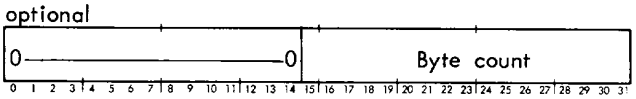
optional



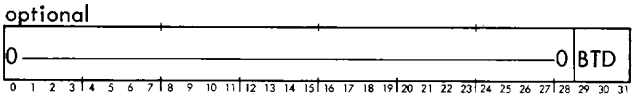
optional



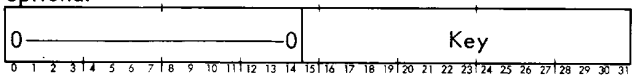
optional



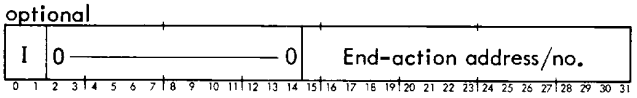
optional



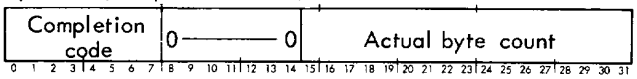
optional



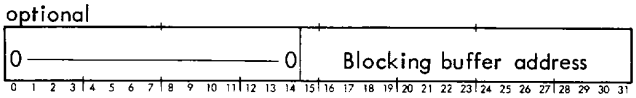
optional



optional (Completion Status)



optional



where

Word 0

Code is X'10' for READ and X'11' for WRITE.

DCB address is the address of the associated DCB.

Word 1

P_1 is the error address parameter presence indicator (0 means absent; 1 means present).

P_2 is the abnormal address parameter presence indicator (0 means absent; 1 means present).

P_3 is the buffer address parameter presence indicator (0 means absent; 1 means present).

P_4 is the byte count parameter presence indicator (0 means absent; 1 means present).

P_6 is the byte displacement parameter presence indicator (0 means absent; 1 means present).

P_8 is the key parameter presence indicator (0 means absent; 1 means present).

P_9 is the end-action parameter presence indicator (0 means absent; 1 means present).

P_{10} is the request type indicator (1 means Type II; 0 means Type I) and indicates the presence of the Completion Status Parameter.

P_{11} is the blocking buffer address parameter presence indicator (0 means absent; 1 means present).

F_1 is the direction indicator for READ (0 means forward; 1 means reverse). This indicator has effect on Magnetic Tape Read/Write operations only.

F_2 is the wait indicator (0 means no-wait; 1 means wait for I/O completion).

F_3 is the RAD Check-Write indicator (1 means write on a RAD will be performed by a Write, Check-Write; 0 means a normal write will be done).

F_4 is the paper tape direct Read/Write indicator (1 means binary paper tape Read/Write operations are performed ignoring the standard paper tape binary record control information; 0 means standard binary format is assumed).

F_5 is the paper tape Read immediate indicator. If F_4 is 1 and F_5 is 1 for a Paper Tape Read, the read is performed without ignoring leading blank frames. If F_4 is 1 and F_5 is 0, paper tape reads are performed ignoring leading blank frames. This indicator is only significant for binary paper tape read requests with $F_4 = 1$.

Word Options

Error address is the address of the entry to the user routine that will handle error conditions for requests specifying wait.

Abnormal address is the address of the entry to the user routine that will handle abnormal conditions for requests specifying wait.

Buffer address is the word address of the user buffer to be used in the I/O operation. Data is written from or read into this buffer. If this parameter is omitted, the buffer address is taken from the DCB (BUF).

Byte count is the size in bytes of the data record. If this parameter is omitted, the record size is taken from the DCB (RSZ parameter).

BTD is the byte displacement (0-3) from the word boundary of the beginning of the data record. If this parameter is omitted and the Buffer address parameter is included in the FPT, value 0 is assumed for BTD. If both parameters are omitted from the FPT, the values of the DCB are used for both.

Key is the number of the granule in a RAD file to be accessed directly. Presence of this parameter implies direct access to a RAD file.

I, End-action address/no. I indicates the contents of the End-action address number field. End-action is allowed only for foreground.

Value 0 indicates an end-action address.

Value 1 indicates an interrupt number.

Value 2 indicates an interrupt operational label.

End-action is taken only in the case of an actual I/O operation involving data transfer to a peripheral.

Completion status is the word wherein the I/O system posts the completion parameters for the request (presence of this parameter indicates that the request is of Type II). The I/O system initializes the word to zero before starting the operation. At completion of the request (cleanup), the actual byte count and the completion code are posted in the word. CHECK may be used to test the parameters.

Blocking buffer address is the address of a 257-word buffer to be used for file directory search if the DCB is being opened to a RAD file.

REWIND, UNLOAD, AND WRITE EOF FUNCTIONS

REW The Rewind causes a data file to be positioned at its beginning if the file is on magnetic tape or a RAD file. Rewind of a file on magnetic tape is accomplished by causing the tape drive to rewind to beginning of tape. Rewind of a RAD file is accomplished by setting the file position parameter in the RAD File Table (RFT) so that the next sequential access request on the file results in the first record being accessed. A Rewind request for a data file on any other device results in no action being taken.

UNLOAD The Unload request results in the same action as Rewind except that magnetic tapes are rewound "off-line". When the rewind is concluded, the user must give an ATTENTION interrupt before the device can be used again. Failure to do so causes the device to time out, and the operator must respond with a key-in before the device can be used again.

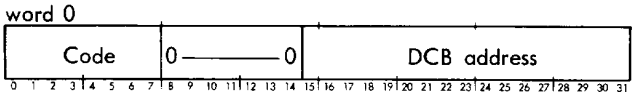
WEOF Write End-of-File causes an EOF to be written if the addressed DCB is assigned to a magnetic tape unit. If the DCB is assigned to a Paper Tape Punch, an ! EOD record is output. If the DCB is assigned to a RAD file, an implicit EOF is written. If the DCB is assigned to any other type of device, no action is taken.

Rewind (REW), Unload (UNLOAD) and Write End-of-File (WEOF) calls are of the form

```
CAL1,1 address
```

where address points to word 0 of the FPT below.

word 0



where

Code is X'01' for REWIND, X'02' for WEOF, and X'03' for UNLOAD.

DCB address is the address of the associated DCB.

FILE AND RECORD POSITIONING FUNCTIONS

These functions are used to alter position within a data file on magnetic tape or RAD.

PFIL, PREC A Position File (PFIL) call causes a magnetic tape to be positioned at the beginning or end of the current file if backward or forward direction, respectively, is specified and no skip is requested. If skip is requested, the tape is positioned as above except that the file mark is skipped over in the specified direction. Position File forward without skip positions the tape at the end of the current file (before the EOF). With skip, Position File forward positions the tape at the beginning of the next file.

Position File causes a "rewind" of RAD files when "backward" is specified, and positioning after the last record in a RAD file when "forward" is specified.

Position Record (PREC) causes a tape or RAD file to be moved n records in the specified direction.

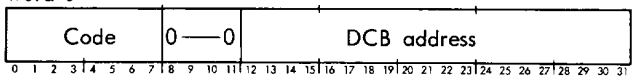
Position File and Position Record are ignored when the data files are on devices other than magnetic tape or RAD file.

File and Record positioning calls are of the form

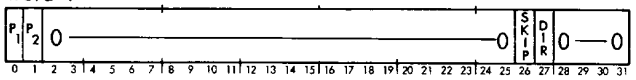
CAL1, 1 address

where address points to word 0 of the FPT shown below.

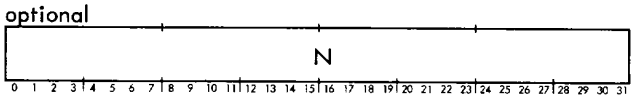
word 0



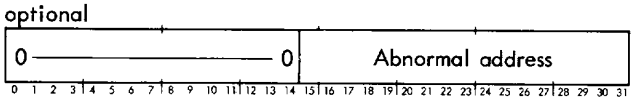
word 1



optional



optional



where

Word 0

Code is X'1C' for Position File, and X'1D' for Position Record

DCB address is the address of the associated DCB.

Word 1

P_1 is the record count (N) address parameter presence indicator (0 means absent; 1 means present). P_1 must be 0 for Position File.

P_2 is the abnormal address parameter presence indicator (0 means absent; 1 means present). P_2 must be 0 for Position File.

SKIP indicates whether the EOF is to be skipped over in positioning magnetic tape. This parameter has significance only for Position File and on magnetic tape (0 means no skip; 1 means skip).

DIR is the direction indicator (0 means forward positioning; 1 means backward positioning).

Word Options

N is the number of records to position.

Abnormal address is the address of the entry to the user's routine that will handle abnormal conditions (EOT, BOT, etc.), for this I/O operation (Position Record only).

PRINT AND TYPE FUNCTIONS

PRINT, TYPE The PRINT function causes the Monitor to list the user's message on the listing log device (operational label LL). The TYPE function causes the Monitor to list the user's message on the operator console device (operational label OC). These functions are reentrant and available to foreground programs. Error and abnormal conditions resulting from these functions are ignored.

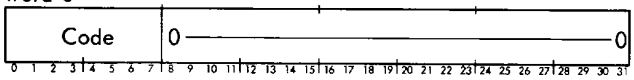
Print and Type may be performed without a wait for completion, but the user is warned that changing the output buffer after return from such a request may result in the output message being modified.

Calls for these functions are of the form

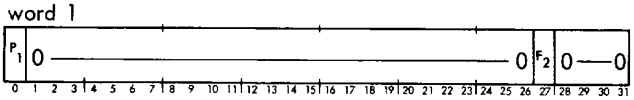
CAL1, 2 address

where address points to word 0 of the FPT shown below.

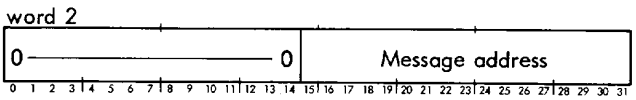
word 0



word 1



word 2



where

Code is X'01' for Print and X'02' for Type.

P_1 is the message address parameter presence indicator ($P_1 = 1$). P_1 is assumed to be a 1.

F_2 is the wait indicator (0 means no-wait; 1 means wait for I/O completion).

Message address is the address of the first word of the message.

Note that the first byte of the first word of the message must specify the number of characters to be listed, up to a maximum of 132 characters for a line printer and 85 characters for a typewriter.

DEVICE/FILE MODE AND FORMAT CONTROL FUNCTIONS

The DEVICE/FILE Mode function is used to set the following parameters:

Modes (MOD, PACK) in the addressed DCB.

Record Size (RSZ) in the addressed DCB and in the RFT entry if the DCB is assigned to an output file.

File Organization in the assigned file's RFT entry if the DCB is assigned to an output file.

Granule Size in the assigned file's RFT entry if the DCB is assigned to an output file.

The parameters set in the RFT entries for permanent files will be written into the RAD file directory entry for the file when the file is closed. Thus, this function defines the parameters for permanent RAD files.

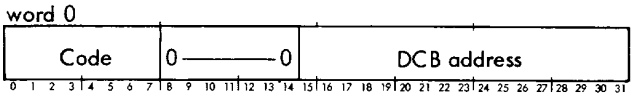
The Device Vertical Format function causes the Monitor to set the vertical-format-control indicator of the specified DCB to 1 or to 0.

Calls for these functions are of the form

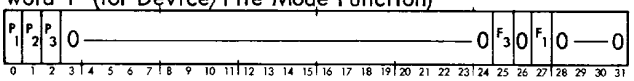
CAL1, 1 address

where address points to word 0 of the FPT below.

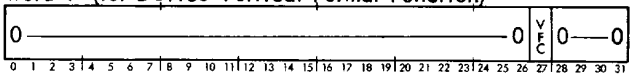
word 0



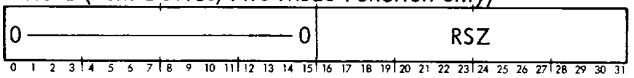
word 1† (for Device/File Mode Function)



word 1† (for Device Vertical Format Function)

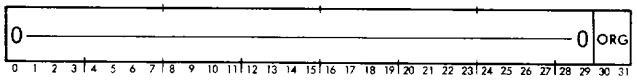


word 2 (with Device/File Mode Function only)

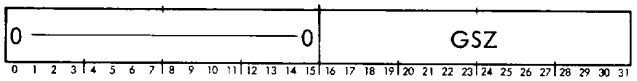


†Two alternative forms of word 1 are shown.

optional



optional



where

Word 0

Code is X'22' for Device/File Mode, X'05' for Device Vertical Format.

DCB address is the address of the associated DCB.

Word 1 (option 1)

P_1 is the record size parameter presence indicator (0 means absent; 1 means present).

P_2 is the file organization parameter presence indicator (0 means absent; 1 means present).

P_3 is the granule size parameter presence indicator (0 means absent; 1 means present).

F_1 1 means BIN; 0 means BCD.

F_3 1 means unpacked format; 0 means packed.

Word 1 (option 2)

VFC is the vertical-format-control specification (0 means no format control; 1 means format control).

Word 2

RSZ is the maximum record size specification, in bytes.

Word Options

ORG is the file organization type:

- 00 for unblocked
- 01 for blocked
- 10 for compress

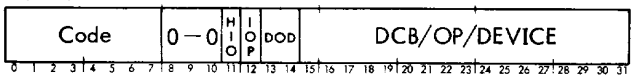
GSZ is the granule size in bytes.

STOPIO, STARTIO These calls are of the form

CAL1, 5 address

where address points to word 0 of the FPT shown below.

word 0



where

Code = X'10' for STOP all system I/O
= X'11' for START all system I/O
= X'0E' for STOP background I/O
= X'0F' for START Background I/O

HIO = 0 for no HIO
= 1 for HIO

IOP = 0 reserve device and controller
= 1 reserve entire IOP

DOD = 00 DCB address is given
= 01 operational label index is given
= 10 device index is given

DCB/OP/DEVICE contains the DCB address, operational label index, or device index as specified.

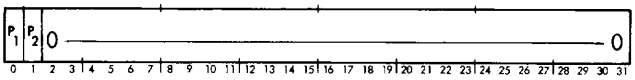
IOEX IOEX calls are of the form

CAL1, 5 address

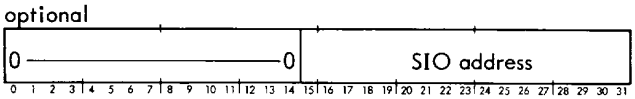
where address points to word 0 of the FPT shown below.

word 0

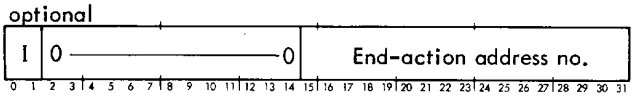
word 1



optional



optional



where

Word 0

Code = X'12' for SIO
= X'13' for TIO
= X'14' for TDV
= X'15' for HIO

DOD = 00 if DCB address is given
= 01 if an operational label index is given
= 10 if a device index is given

DCB/OP/DEVICE contains the DCB address, operational label index, or device index as specified by DOD.

Word 1

P_1 is the SIO address parameter presence bit (0 means absent; 1 means present).

P_2 is the end-action parameter presence bit (0 means absent; 1 means present).

Word Options

SIO address is the address of the IOP command doubleword

I, End-action address no. I indicates the contents of the End-action address number field.

Value 0 indicates end-action address.

Value 1 indicates interrupt number.

Value 2 indicates interrupt label.

The TIO, TDV, and HIO operations are performed immediately and the condition codes and status are returned as shown in Table 13.

The command pairs for an SIO operation are not checked for validity. The flags in the command pairs may be set according to the needs of the user. The SIO is issued whether or not the device is busy or in manual mode, and the status is returned to the caller. It is the user's responsibility to sense for the manual mode before and after the SIO request and then inform the operator with a suitable message.

When I/O interrupts occur as a result of IOEX (SIO only), end-action is initiated as requested in the FPT. The end-action is identical to that for READ/WRITE calls.

Table 13 shows the status returned from the different IOEX functions.

Table 13. IOEX Function Status Returns

Operation	Major Status	Condition Codes	Register 8	Register 9
ALL	Device not preempted	0001	- - -	- - -
ALL	No I/O address recognition	1100	- - -	- - -
SIO	I/O address recognized and SIO accepted	0000	Current command address	Status and byte count
	I/O address recognized but SIO not accepted	0100	Current command	Status and byte count
	Device controller is attached to "busy" selector IOP	1000	- - -	- - -

Table 13. IOEX Function Status Returns (cont.)

Operation	Major Status	Condition Codes	Register 8	Register 9
TIO	I/O address recognized and SIO is currently possible	0000	Last command address	Status and byte count
	I/O address recognized but SIO not possible	0100	Current command address	Status and byte count
	Device controller attached to "busy" selector IOP	1000	- - -	- - -
TDV	I/O address recognized	0000	Current command	Status and byte count
	I/O address recognized and device dependent condition present	0100	Current command address	Status and byte count
	Device controller attached to "busy" selector IOP	1000	- - -	- - -
HIO	I/O address recognized and device controller is not busy	0000	Current command	Status and byte count
	I/O address recognized but device controller "busy"	0100	Current command address	Status and byte count

5. USER PROGRAM SCHEDULING AND OPERATION

SCHEDULING AND LOADING PROGRAMS

The Overlay Loader links relocatable object modules to form an absolute load module representation of the program. The load module is created as a RAD file and consists of a header and an absolute core image of the various program segments. The load module header contains the program parameters used by the Root Loader for Loading the program root.

LOADING AND RELEASING FOREGROUND PROGRAMS

Loading and initializing of a foreground program root is performed by the Foreground Root Loader, and involves the following steps:

1. Opening the file containing the absolute load module.

2. Building a Foreground Program Table entry that contains the program name, core memory to be used by the program (root and all segments), and the public libraries used by the program. The last two parameters are taken from the load module header.

3. Testing for required core size availability. If some portion of required foreground memory is busy, a message is typed, the Foreground Program Table entry is purged, and an error return is given. If no busy foreground core is required but some portion of background is needed, the background is checkpointed and the core area marked as foreground. If the foreground memory is not busy or if the background has been checkpointed (if necessary), the load process

   - Loads the program root.
   - Sets the PCB pointer in X'4E' to point to the foreground user program PCB.
   - Transfers control to the start address, taken from the load module header, where the user program initializes itself (connects tasks, conditions interrupts, etc.).

When initialization is completed, the user program performs an EXIT function call. The EXIT function will recognize that initialization of a foreground program has been completed and will transfer control back to the RBM Control Task (the EXIT call does not cause an exit from the RBM Control Task).

A foreground program root can be loaded by any of the following:

! RUN control command

! ROV control command

RUN key-in

RUN system call from a foreground task

Since the root loading occurs at the level of the RBM Control Task, foreground programs making RUN calls must give up the CPU (EXIT) before the load can be accomplished. Foreground tasks can request the triggering of an interrupt at conclusion of the root load and initialization.

Release of foreground programs is also performed at the RBM Control Task priority through the following steps:

1. Disarming all interrupts connected to tasks within the program. The interrupts are specified in the INTTAB which is pointed to by a word in the program PCB.

2. Closing any open DCBs within the program to cause I/O run-down in addition to closing data files.

3. Purging the foreground program entry, which has the effect of marking the memory as not busy.

4. Restarting the background if it has been checkpointed, and if release of this foreground program makes available the memory needed for the background.

Release of a foreground program occurs as a result of either an RLS key-in or RLS system call.

LOADING AND EXECUTING BACKGROUND PROGRAMS

The Background Root Loader loads background programs as specified by control commands in the background job stream at the Control Task priority level.

The Background Root Loader will only load a background program root if the background memory area is large enough to contain the entire program (root and segments). Upon completion of the root load, control is transferred to a background program at its start address. The background program terminates execution with an EXIT or ABORT return. After terminating a background program, RBM resumes processing the control commands from the background job stream.

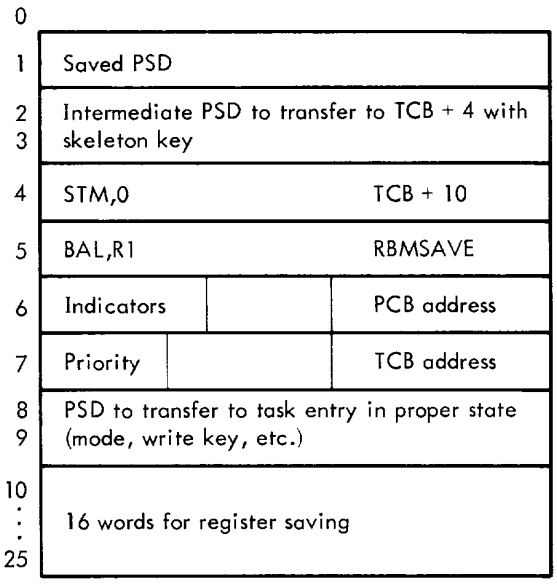
TASK CONTROL BLOCK (TCB)

A Task Control Block must be associated with each centrally connected real-time task, and is used by the system to save the context of the interrupted task upon occurrence of the given task's interrupt. The TCB is in the user program (assembly language users must allocate and define their TCBs in the source code of their program). The FORTRAN compiler generates implicitly the TCBs needed for a real-time FORTRAN program.

TASK CONTROL BLOCK FORMAT

The assembly language user must allocate a TCB in the source code for each centrally connected task in the program. Each TCB begins on a doubleword boundary and has a length of 26 words.

The RBM CONNECT function fills in the TCB. When completed, a TCB has the following form:



where RBMSAVE is a system routine that

1. Exchanges the contents of X'4E', X'4F', and TCB +6,7.
2. Sets appropriate indicators in TCB +6.
3. Transfers to the task starting address by LPSD from TCB +8.

Users must never alter any portion of a TCB.

PROGRAM CONTROL BLOCK (PCB)

The Program Control Block contains the program-associated parameters used by the RBM system to provide service functions for the program. Every program, background and foreground, contains a PCB that is allocated and constructed by the Overlay Loader.

The Program Control Block defines the program Temp Stack to be used by a program. It also contains a pointer to a list of the DCBs associated with a program and a pointer to a list of interrupts connected to the tasks within a foreground program. This table of interrupts is filled by the CONNECT routine as the various connect calls are made.

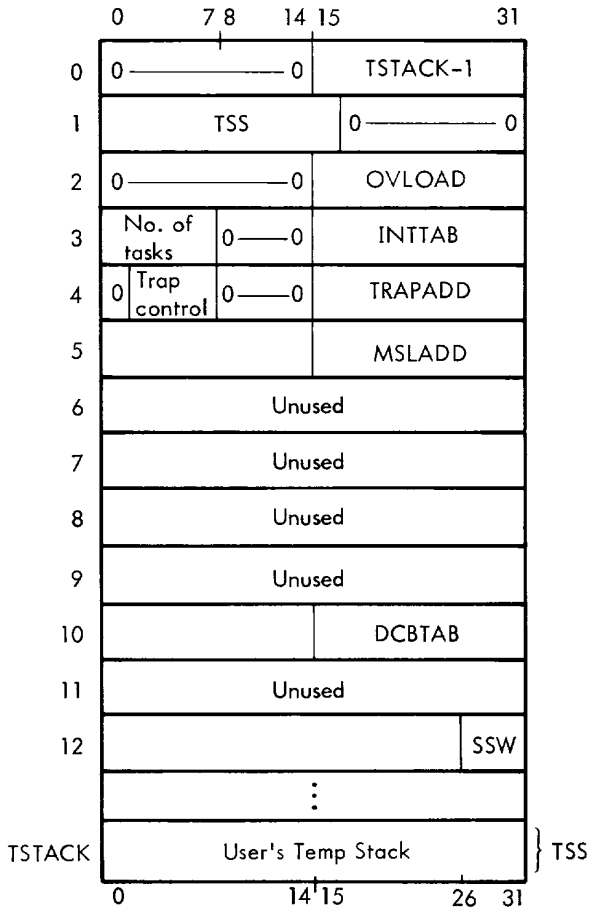
Since the Temp Stack is associated with the program rather than individual tasks, different tasks within a program should not use these stacks for data communication. Common storage can be used for communication between tasks or between occurrences of a given task.

At all times during operation of RBM, location X'4E' and X'4F' contain pointers to the PCB and TCB, respectively. In addition, X'4F' contains the current priority level in byte 0.

PCB FORMAT

The PCB is built by the Overlay Loader from parameters specified on the !OLOAD control command.

The PCB is of the form



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.

OVLOAD is the address of the table used by the segment loader to manage the program overlays.

No. of Tasks is the number of tasks in the program. This is also the number of entries in INTTAB.

INTTAB is the address of the interrupt table associated with the program. This table is maintained by the CONNECT function. The format of this table is shown below.

Trap Control (Bits 1-7) specify how the various traps are to be handled. An explanation of these bits is given in the TRAP function description later in this chapter.

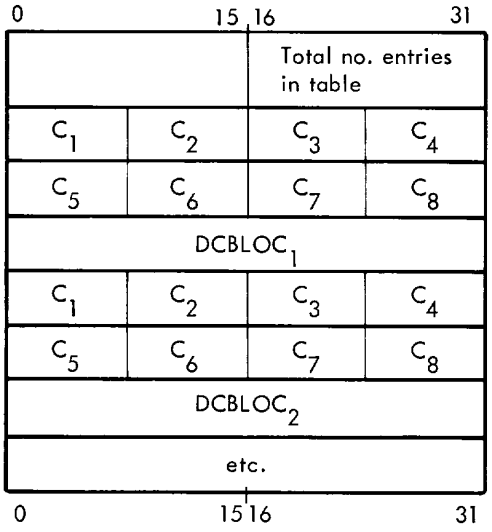
TRAPADD is the address of the user's routine which processes the various traps.

MSLADD is the address of the M:SL DCB, which is used to load overlay segments.

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.

SSW contains the user's sense switch settings. Bit 26 contains the setting of switch 1, etc. The sense switches have no use in this initial release of RBM.

DCBTAB Format

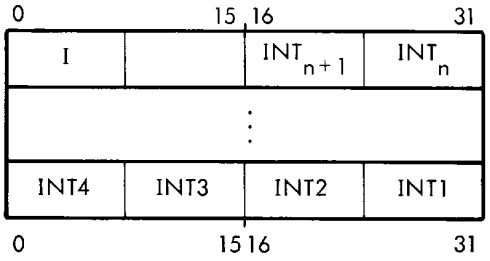


where

C_1-C_8 indicates the EBCDIC name of the DCB, left-justified, with trailing blanks.

DCBLOC is the absolute address of the first word location of the DCB.

INTTAB Format



where I is the index value used to access the next available entry in the table $0 < I \leq 4N - 1$. N is the number of words allocated by the Loader for the table. The table is maintained by the CONNECT system call. I has an initial value equal to the number of tasks.

Each byte (interrupts 1 to n) represents the priority of the interrupt where a value of 1 represents the highest priority and corresponds to interrupt location X'50'.

TEMP STACK

The Temp Stack is a "push-down/pull-up" stack of memory locations that have been allocated by the Overlay Loader. It is required for Monitor functions and subroutines that use Temp Stack storage (i.e., FORTRAN IV-H Library routines).

The user can manipulate the Temp Stack by push/pull stack instructions (PSW, PLW, PSM, PLM, MSP) that indirectly address location '4E', which contains the address of the executing program's PCB. The first doubleword of the Program Control Block is the stack pointer doubleword used in allocating (pushing) and releasing (pulling) blocks within the Temp Stack.

The "push-down/pull-up" functions operate on a last-in, first-out basis, and these operations must be symmetrical in number and size. An attempt to push a block that is greater than the remaining stack space results in overflow. Similarly, an attempt to pull more out of the Temp Stack than had been previously pushed down would result in underflow. These conditions result in traps that may be handled by the user (see TRAP system call).

The size of the Temp Stack must be equal to or greater than the total number of temp cells required by the maximum number of nested routines using temporary storage; (i.e., if a Monitor I/O routine needs 16 temp cells and it calls a routine that needs 19 cells, the total number of cells required would be 35). The number of cells required for system CALs is 18 to 70; the FORTRAN IV-H Library subroutines require 148 temp cells for each task.

Foreground tasks at different interrupt levels within the same program share the program's Temp Stack, and allocation must be sufficient to accommodate the maximum number of tasks that could be enabled at one time. When an executing task exits, it must restore the temp stack pointer to its original condition. This is particularly important in a foreground program where a Temp Stack is allocated for each program and not each task. Thus, if several tasks in the same program share the program's Temp Stack, the housekeeping of the Temp Stack pointer (e.g., symmetrical pushes and pulls) must be meticulous.

MASTER AND SLAVE MODES

Foreground tasks can change their execution mode through MASTER and SLAVE system function calls. At entry to a task, the mode is set as specified by the function call that caused the connection.

Note: Serious consequences can result from improper operation in master mode.

OVERLAY SEGMENT LOADING

Overlay segments are loaded through a SEGLOAD system function call and the segment to be loaded is specified by number.

Both background and foreground programs may load overlay segments. The memory space necessary for a foreground program overlay is always available, since availability was verified when the root was loaded, and the total space used by the program and all its overlays was marked as busy by the Root Loader. Overlay segments are loaded at the priority level of the requesting task.

CHECKPOINT AND RESTART

Checkpoint and restart of the background are performed automatically by the system as foreground programs are loaded and released. Checkpoints are performed as necessary to load foreground programs. Restarts are performed as possible when foreground programs are released. Both functions are performed in the RBM Control Task. Checkpointing consists of

1. Writing the background program to a fixed area on the RAD (specified at SYSGEN) after the background program has stabilized (no background I/O request running).
2. Marking the background memory as unused foreground to enable loading of foreground programs into that area.

Restarting consists of

1. Reading the checkpointed background program from the RAD into memory if the required memory area is marked as unused foreground (no active foreground programs in the area), and marking this area as background.
2. Restarting the background program at the point it was interrupted by the checkpoint request.

TRAP HANDLING

RBM provides standard processing of trap conditions. A user can either take advantage of the system processing or request that he himself handle certain trap conditions. Also, certain traps can be ignored. System trap handling involves aborting (for background) or exiting (for foreground) the task that is active at the occurrence of a trap, with the following exceptions:

1. An unimplemented instruction trap occurrence will result in the instruction being simulated if the simulation package is in the system. If simulation is impossible, the program will be aborted or exited.

2. The user can mask out fixed-point arithmetic and decimal arithmetic traps either through system call (TRAP function) or at task connection time (CONNECT, ARM, DISARM functions).

3. The user can mask out some floating-point trap occurrences through use of the LCF and LCFI instruction.

4. Any unmasked trap can be received by the active program. This is set up through the TRAP function call, wherein the user can specify the address of a routine to handle the various trap conditions. This address is kept on a per program basis as opposed to a per task basis.

When the user program receives control in the trap routine, the following items are stored in sequence in a 19-word block of the program's Temp Stack, starting on a doubleword boundary: the PSD and the 16 registers saved when the trap occurred, and a word containing the trap location (right-justified). Register 1 points to word 0 (first word of the PSD) in this block. If 19 words are not available in the Temp Stack, a background program is aborted; a foreground task is exited.

The address of the user trap routine and the control bit for each trap is kept in the PCB.

User trap handling or system trap handling is also invoked if an invalid parameter exists in an FPT for a system call that is unable to post the error condition. In this case, an error code of X'50' will be posted in the last word of the user's Temp Stack if a user trap address is present in the PCB.

Return from the user trap routine to the interrupted program is accomplished by the TRTN function, which restores the context from the Temp Stack and returns to the location following the trapped instruction.

RETURN FUNCTIONS

RBM provides users with the following return functions:

EXIT is used by background programs at the normal completion of a background program. EXIT is used by foreground programs when a centrally connected task has concluded the processing of an interrupt. An EXIT call from the foreground causes the system to restore the context of the interrupted program and return to the point of interrupt.

ABORT is used by background programs to cause the system to abort the job containing the program. The system will abort the background program and type a message on the operator console (OC) that the job was aborted and give the address of the ABORT call unless an !ATTEND command was present in the background job. The system will read and ignore all records from the C device until the next ! JOB card is encountered. If an ABORT call is made from a foreground program, an EXIT is performed.

INTERRUPT CONTROL

CONNECTING TASKS TO INTERRUPTS

Interrupt connection may be accomplished through use of the CONNECT, ARM, or DISARM function calls. While these calls are usually made during foreground program initialization (performed at the Control Task priority level), this is not a requirement. A table of interrupts connected within a program is kept by the system. This table is pointed to by an entry in the program's PCB, and space for the table is allocated by the Overlay Loader in the user program. The table enables the foreground program release function to

disarm and disable all interrupts associated with a foreground program. In calling for connection the user may specify the mode (master, slave) and the interrupt inhibit conditions that are to exist at entry to the specified task. Two types of connections are available, direct and central:

1. Direct connection results in immediate entry to the task upon occurrence of the interrupt. The task must ensure that the context is saved, as necessary, and restored upon exit. The task should set the priority field (byte 0) in location X'4F' to ensure that CPU time is not stolen for I/O cleanup for lower-priority tasks. If the task uses any RBM services, locations X'4E' and X'4F' must be properly set.
2. Central connection results in entry to the task after the interrupted task context has been saved. The CONNECT function constructs the TCB so that the context save will occur. Exit from a centrally connected task is by EXIT, which will restore the interrupted task status and return to the interrupted task.

   To perform the connection, the system fills in the TCB previously shown, The PSD is constructed (TCB + 8) to transfer control to the user in the proper mode and with the proper write key. An XPSD TCB instruction is placed in the proper interrupt location to complete the connection.

The CONNECT function also notes in the INTTAB the number of the interrupt being connected so that the interrupt can be disarmed and disabled when the foreground program is released. Details of the system calls concerned with connection are given later in this chapter under "System Function Call Formats".

ARMING, DISARMING, ENABLING, DISABLING

These functions can be performed by the ARM, DISARM, ENABLE, and DISABLE system function calls, which specify an interrupt by number or label. As options on ARM and DISARM, connection of the interrupt to a task can be performed, and/or a clock can be set to interrupt at specific intervals.

ARM, DISARM, ENABLE, and DISABLE functions can also be performed by operator request through the CINT key-in.

TRIGGERING OF INTERRUPTS

An interrupt can be triggered through a TRIGGER system function call. The interrupt to be triggered is specified by number or by label. TRIGGER calls may be made from any foreground task. An interrupt can also be triggered by operator request through the CINT key-in.

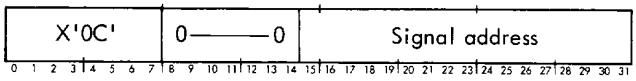
SYSTEM FUNCTION CALL FORMATS

RUN This function call is available only to the foreground. It has the format

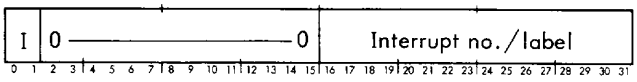
CAL1,5 address

where address points to word 0 of the FPT show below.

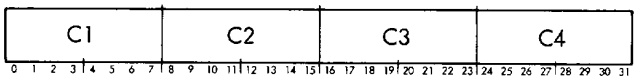
word 0



word 1



word 2



word 3

where

Word 0

X'0C' is the code for the RUN call.

Signal address is the address of a status word into which the system posts one of the following signals:

- 0 if the program was successfully loaded.
- 1 if the space was not available in the foreground memory area or if there was insufficient space in the Foreground Program (FP) table to make entries for the public libraries needed by the program.
- 2 if the requested program did not exist on the FP area of the RAD or if an I/O error occurred attempting to load the program.
- 3 if the program is already loaded.
- 4 if a previous request has been made to load the same program but the program is not yet loaded. In this case, the Foreground Root Loader is able to notify only the first requester when the program is loaded.
- 5 if the space was not available in the Foreground Program (FP) table for the requested program.
- 6 if invalid attempt has been made to load a public library. Since public libraries are automatically loaded and released by the Foreground Root Loader, they cannot be loaded via a RUN call.

The signal address cannot be a register (addresses 0-F) and must be in the calling program's portion of memory. An invalid signal address results in control being transferred to the System or User Trap Handler.

The user should inspect the signal word upon return from the CAL, since some of the signals (3, 4, and 5) are posted immediately by the RUN processor. The signal word should be initialized to a value other than 0-6, so that the user can determine if or when the signal is posted.

Note that an interrupt is triggered only if the RUN request is passed on to the Foreground Root Loader. That is, signals 3, 4, and 5 are returned immediately by the RUN processor and, in this case, no interrupt is triggered.

Alarms are output by the Foreground Root Loader for error conditions 1, 2, and 6.

Word 1

I, Interrupt no./label indicates the contents of the Interrupt number/label field (0 for no interrupts; 1 for interrupt number; 2 for interrupt label).

This interrupt is triggered by the Root Loader at the conclusion (successful or unsuccessful) of the root load and initialization.

Words 2-3

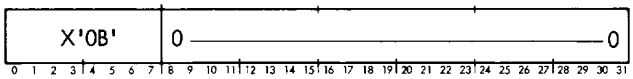
Ci are the characters in the name of the load module. The name is left-justified with trailing blanks.

RLS This function call (Release Foreground Program) is available only to the foreground, and has the format

CAL1,5 address

where address points to word 0 of the FPT shown below.

word 0



word 1

word 2

where

X'0B' is the code for the RLS call.

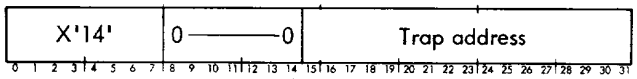
Ci are the characters specifying the name of the program. The name is left-justified and filled with trailing blanks. An invalid name results in a return with no action taken by the system.

TRAP This function call has the form

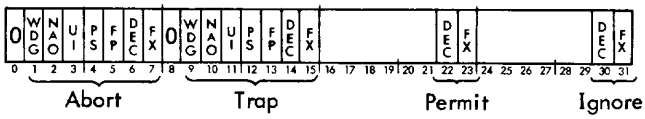
CAL1,8 address

where address points to word 0 of the FPT shown below.

word 0



word 1



where

Word 0

X'14' is the code for the TRAP call.

Trap address is the address in the user program that receives the requested traps. The address is optional unless it is the initial call and one of the trap bits is set. The address must lie in the calling program's portion of memory.

Word 1

Bits 1-7 are the Abort flags specifying which traps are to be handled by the System.

Bits 9-15 are the Trap flags specifying which traps are to be handled by the user's trap handler.

Bits 22-23 are the Permit flags specifying that the decimal or arithmetic mask in the PSD is to be set so that these traps are permitted.

Bits 30-31 are the Ignore flags specifying that the decimal or arithmetic mask in the PSD is to be set so that these traps are ignored.

The Abort, Trap, Permit, and Ignore fields specify the changes to be made in the disposition of trap occurrences.

The bits in these fields have the following significance:

WDG	Watchdog Timer
NAO	nonallowed operation
UI	unimplemented instruction
PS	pushdown stack limit
FP	floating-point arithmetic
DEC	decimal arithmetic
FX	fixed-point arithmetic

If a control bit has value 1, the trap is to be handled as specified. A value of zero specifies that no change is to be made in the handling of that trap. The fields are processed from left to right (Abort, Trap, Permit, Ignore), with the last-processed code overriding any previously processed code.

If a given trap condition has a control bit value of 1 in both the Abort and Trap fields, the Trap bit will override the Abort bit and the user will receive the trap, since the Trap bit is the last one processed.

TRTN This function call (trap return) is of the form

```
CAL1,9    5
```

No FPT is used.

EXIT This function call is of the form

```
CAL1,9    1
```

No FPT is used.

ABORT This function call is of the form

```
CAL1,9    3
```

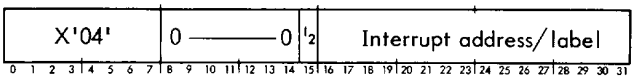
No FPT is used.

CONNECT This function call (available only to the foreground) has the form

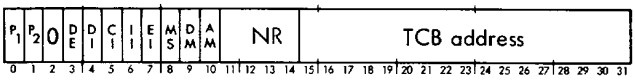
```
CAL1,5 address
```

where address points to word 0 of the FPT shown below.

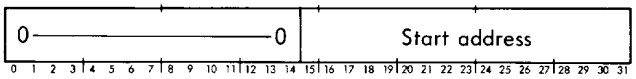
word 0



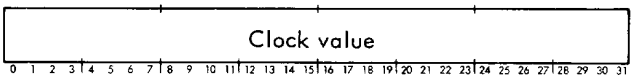
word 1



optional



optional



where

Word 0

X'04' is the code for the CONNECT call.

I_2 indicates that the address ($I_2 = 0$), or the label ($I_2 = 1$), of the interrupt is specified in Interrupt address/label.

Word 1

P_1 is the task start address parameter presence indicator (0 means absent; 1 means present). This indicator must be 1.

P_2 is the clock value parameter presence indicator (0 means absent; 1 means present).

DE specifies that the interrupt is to be disabled (DE = 1), or enabled (DE = 0).

DI specifies that the connection is to be direct (DI = 1), or central (DI = 0).

CI specifies that the task is to be entered with the clock group inhibit set (CI = 1), or reset (CI = 0).

II specifies that the task is to be entered with the I/O group inhibit set (II = 1), or reset (II = 0).

EI specifies that the task is to be entered with the external group inhibit set (EI = 1), or reset (EI = 0).

MS specifies that the task is to be entered in master mode (MS = 0), or slave mode (MS = 1).

DM specifies that the task is to be entered with the decimal mask set (DM = 1), or reset (DM = 0).

AM specifies that the task is to be entered with the arithmetic mask set (AM = 1), or reset (AM = 0).

NR is the number of registers to be saved upon occurrence of the interrupt (if connection is central). Value 0 is used to denote that 16 registers are to be saved. Registers are saved beginning with register 0, and at least four registers must be saved.

TCB address contains the TCB address for central connection. For direct connection, this portion of word 1 is unused.

Word Options

Start address is the starting address of the task if it is to be centrally connected. If the task is to be directly connected, this is the address of the XPSD to be executed in the interrupt location. The user-furnished XPSD instruction will be stored in the task's interrupt location by the CONNECT function.

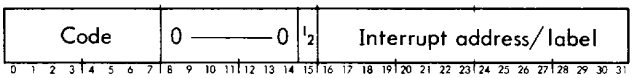
Clock value is the value (in units of the clock's resolution) to which the clock is to be set. If this parameter is absent, no change is made in the interval. This value should be presented if a task is being centrally connected to a clock interrupt. Note that the clock value is decremented via an MTW,-1 instruction. This parameter is meaningless for a direct connection or if no connection is being made.

ARM, DISARM These function calls (available only to the foreground) are of the form

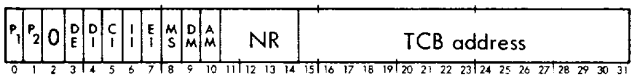
CAL1,5 address

where address points to word 0 of the FPT shown below.

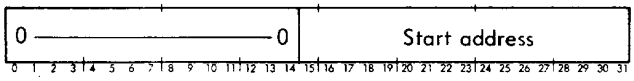
word 0



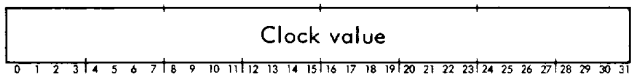
word 1



optional



optional



where

Code = X'03' specifies DISARM
= X'04' specifies ARM

I_2 has the same significance as in the CONNECT call.

If $P_1 = 1$, a connection is performed and the parameters in word 1 and the optional words assume the same significance as in the CONNECT call.

If $P_1 = 0$, no connection is performed and the remainder of the parameters in word 1 are ignored, except for the DE parameter in the case of an ARM call.

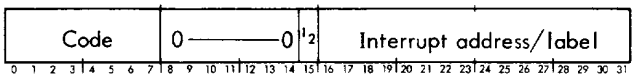
The rest of the coding is identical to that for the CONNECT function call.

ENABLE, DISABLE, TRIGGER These function calls (available only to the foreground) are of the form

CAL1,5 address

where address points to word 0 of the FPT shown below.

word 0



where

Code = X'00' specifies TRIGGER
= X'01' specifies DISABLE
= X'02' specifies ENABLE

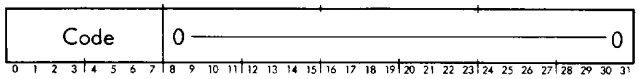
I_2 specifies that either the address ($I_2 = 0$), or the label ($I_2 = 1$), of the interrupt is specified in Interrupt address/label.

MASTER, SLAVE These function calls (available only to the foreground) are of the form

CAL1,5 address

where address points to word 0 of the FPT shown below.

word 0



where

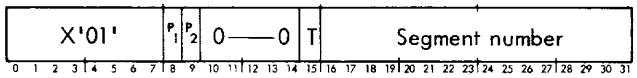
Code = X'07' specifies SLAVE
= X'08' specifies MASTER

SEGLOAD This function call is of the form

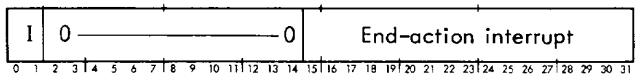
CAL1,8 address

where address points to word 0 of the FPT shown below.

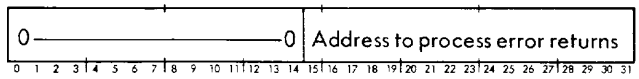
word 0



optional



optional



where

Word 0

X'01' is the code for the SEGLOAD call.

P_1 indicates the presence or absence of word 1; 0 means absent, 1 means present.

P_2 indicates the presence or absence of word 2.

T indicates whether control is to be returned following the call or transferred to the starting location of the segment at the conclusion of the segment load,

0 for return to calling program.

1 for transfer to new segment (only valid if $P_1 = 0$).

Word Options

I indicates the contents of the end-action interrupt field (foreground only):

I = 0 indicates no end-action.

I = 1 indicates an interrupt address.

I = 2 indicates an interrupt label.

If end-action is specified, the request to load the segment will be queued, and control will be returned immediately to the calling program. The calling program can then exit and release control while the segment is being loaded. If end-action is not specified (I = 0), control will not be returned until the segment is loaded. The user is responsible for checking the status of the load if end-action is selected.

Address to process error returns This is the address of the user's routine for processing any error or abnormal returns received while attempting to load the overlay segment. The codes returned will be identical to those of the READ CAL since a READ CAL is used by SEGLOAD to load the segment. If this address is not present and an error occurs, a foreground program will be exited or a background program aborted. If an error is detected in the user's PCB or OVLOAD table, the User or System Trap Handler will be entered.

WAIT A background program will enter the "wait" state through this function call if an !ATTEND card was included in the control commands for the job. Normally, a background program would use WAIT after typing an alarm to the operator that requires an operator response. While in this state, the Control Task waits for a key-in from the operator specifying the disposition of the background program. The operator may specify continue (C), continue from OC (COC), or abort (X).

This function call is of the form

```
CAL1,9     9
```

No FPT is used.

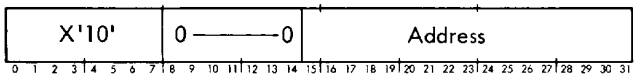
TIME Programs may interrogate the Monitor to determine the time of day and date.

This function call is of the form

```
CAL1,8 address
```

where address points to word 0 of the FPT shown below.

word 0



where

X'10' is the code for the TIME call.

Address is the address of the first word of a 4-word block which is to receive the time and date. In systems without Job Accounting or if a data has never been input via a "DT" key-in, the 4-word block will not be modified. This block contains EBCDIC characters as shown below.

word 0

h	h	:	m

word 1

m	b	m	o

word 2

n	b	d	d

word 3

,	'	y	y

where

hh is the hour ($0 \leq hh \leq 23$).

mm is the minute ($0 \leq mm \leq 59$).

mon is the month (3-letter abbreviation).

dd is the day ($01 \leq dd \leq 31$).

yy is the year ($00 \leq yy \leq 99$).

6. OVERLAY LOADER

OVERVIEW

The Overlay Loader is a two-pass processor that creates programs in overlay form. Modules in Sigma 5/7 Standard Object Language format are converted to overlays in absolute core image form in accordance with the Loader control commands. The Loader creates programs for execution in either foreground or background, prepares standard processors for execution under the Job Control Processor, and creates Public Libraries.

The Overlay Loader permits the user to assemble, load to background or foreground areas, and execute programs with minimal control information. The default cases documented in this section for each control command will handle most normal situations.

The control command structure permits the user to tailor the loading procedure for a wide variety of situations, and the control commands add control and flexibility by overriding default cases and adding options.

The size of the program that can be loaded is a function of the size of the symbol table and available core storage at load time, rather than the amount of core memory that the program occupies at execution time. Therefore, the Overlay Loader may load user programs equal in size to the maximum available area in core at execution time, even though this area is not available at load time.

The loading of mixed media is allowed, and all library loading will be from library files on the RAD. A library need not be ordered.

FUNCTIONAL FLOW

The options specified on the !OLOAD control command are scanned and those not specified are assigned their default values. A :ROOT or :SEG control command is scanned to determine the source of the binary object modules from which the segment will be created, and to define its linkage.

The Loader makes the first pass over the binary object modules, allocating the segment's labelled COMMON blocks (dummy sections) and control sections. It concurrently builds a symbol table of DEFs and unsatisfied REFs. Object modules input from non-RAD devices are saved on a temporary RAD file (X1).

After the last object module for a segment has been input, the Libraries are searched. Pointers to the selected library object modules are saved and their DEFs and REFs are added to the symbol table. At the end of a path, segment symbol tables are written on temporary RAD files. The blank COMMON base is set at the end of the first pass. At this point, all the Loader control commands except :ASSIGN have been input.

During the second pass, each segment's binary object modules and selected library modules are loaded. The absolute core image of each segment is created and written on the program file. Part two of the ROOT (the INTTAB, DCBTAB, OVLOAD table, the Temp Stack, and any DCBs created by the Loader) is built at the end of the second pass. If a MAP has been specified, it is output. If an output file used by the Loader overflows, an attempt is made to output all possible MAP information. The Loader returns to the Monitor by calling either the EXIT or ABORT function.

LIMITATIONS

There are certain limitations in the use of the Overlay Loader due to total system considerations or because the efficiency of the Loader could otherwise be degraded.

1. No discontinuous programs will be output by the Overlay Loader. The Monitor SEGLOAD function reads only a contiguous core image. Since each discontinuity would result in at least one additional RAD access, considerable degradation of the run-load process for the foreground would result.
2. The contents of reserve areas within a program will not be predictable at execution time unless initialized in some manner (e.g., by DATA statements). Labeled COMMON will be unpredictable unless initialized by a DATA statement. Blank COMMON is not written to the RAD and is not loaded as part of the program.
3. Allocation of program, COMMON, and Labeled COMMON within a program area is generally determined by the Loader.
4. Only relocatable modules or those containing absolute origins falling within the limits of the segment currently being loaded will be allowed.
5. No implicit loading of segments will take place at execution time. Only explicit calls to the Monitor SEGLOAD function will read in overlay segments. Thus, the overlay structure must be accurately defined at load time to coincide with explicit calls in the user's program.

OVERLAY PROGRAMS

An overlay program is defined as the collection of absolute core image segments generated by the Overlay Loader. The Loader produces background overlay programs (including processors) and foreground overlay programs. Note that the Overlay Loader loads only programs; a foreground program may consist of one or more tasks.

OVERLAY STRUCTURES

An overlay program is generally composed of a root and several overlay segments; however, it can consist of only a root without overlay segments. The root segment is resident at all times during the execution of a program; overlay

segments are resident only when they have been explicitly read by calls to the Monitor SEGLOAD function.

Each segment is created from one or more binary object modules and associated library routines. The segments are assigned arbitrary identification numbers (except for the root which is always segment 0) that must be unique within the overlay program. Segment numbers are used by the Overlay Loader and the Monitor SEGLOAD function.

The overlay structure is communicated to the Loader with the :ROOT and :SEG control commands. Since each overlay segment is created and stored on the RAD as a continuous string of bytes in absolute core image form, data in reserved areas of the program segment is not predictable. Note that reserved areas and data blocks will effectively be reinitialized each time that an overlay segment is read in by SEGLOAD. The structure formed by segments that can exist in core at any one time is called a path.

The overlay program example given in Figure 3, consists of a root (segment 0) and overlay segments 1 through 15. The segments (horizontal lines) are numbered in the order in which they were built by the Loader. There are nine paths:

1. 0,1,2
2. 0,1,3
3. 0,4
4. 0,5,6,7
5. 0,5,6,8
6. 0,5,9,10,11,12
7. 0,5,9,10,11,13
8. 0,5,9,10,14
9. 0,5,9,15

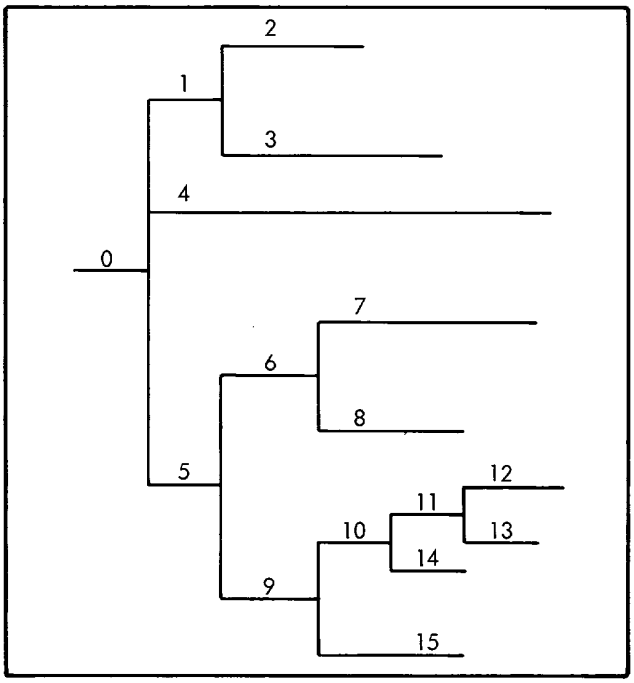


Figure 3. An Overlay Program

OVERLAY RESTRICTIONS

Communication between segments by external DEF/REF linkages is permitted with the following restrictions:

1. The Loader will satisfy a DEF/REF linkage only within a path.
2. A segment in one path cannot reference a segment in another path. For example, segment 2 must not reference any of segments 3-15.
3. The user must ensure that any segments that intercommunicate are in core. For example, if segment 5 references segments 6 and 8, then segments 6 and 8 must have been explicitly loaded. If segment 8 references segments 5 or 6, these segments must have been explicitly loaded since the loading of segment 8 does not cause the implicit loading of segments 5 and 6.
4. Identical definitions cannot be used in segments that are in the same path. For example, segments 5 and 13 cannot have identical definitions because they are both in path (0,5,9,10,11,13).
5. Identical definitions and references may be used in segments of different paths that do not involve a common segment. For example, if segments 7 and 15 reference identical definitions in segments 6 and 9, the Loader will link the reference in 7 with the definition in 6 and the reference in 15 with the definition in 9.
6. Identical references in segments of different paths may be made to a definition in a segment common to both paths. For example, segments 6 and 9 can each reference a definition in segment 5 because 5 is a common segment in the two paths (0,5,6,7) and (0,5,9,10,14).
7. A segment that is common to two paths cannot reference identical definitions in the different paths. For example, segment 10 cannot reference identical definitions in segment 12 and 13, even though segments 12 and 13 are in different paths.

Where possible, the Loader will warn the user about errors in overlay structure and segment communication; however, it is the user's responsibility to attempt a reasonable, workable overlay contruction.

OVERLAY CONTROL COMMANDS

The prime Overlay Loader command, !OLOAD, is read by the Job Control Processor and causes the Overlay Loader processor to be read into the background and executed. All Loader subcommands are identified by a leading colon (e.g., :SEG). They are read from M:C and logged onto M:LL. Blank cards are passed over without comment. When a Monitor control command is encountered, the Loader completes the load process and exits to the Monitor.

Note that !EOD must occur only as a terminator for object module input; its use is illegal for terminating the Loader control command stack.

SYNTAX

The syntax for Overlay Loader control commands is identical to that defined for the RBM Monitor (except for MODIFY control commands).

ORDER OF CONTROL COMMANDS

The control command stack is divided into major divisions or substacks, which must occur in the following order:

```
! OLOAD
:ROOT  }
:SEG   }
  .    }        or        {:PUBLIB}
  .    }
:SEG   }
:ASSIGN}
!  (Monitor control command)
```

The :COMMON, :LCOMMON, :LIB, :INCLUDE, :EXCLUDE, :RES, and :MODIFY control commands may occur in any :ROOT or :SEG substack and apply only to that root or segment. The :ASSIGN control commands must follow all other commands in the stack. The :PUBLIB control command is unique, replacing :ROOT, :SEG, and :ASSIGN substacks.

A ROOT or SEG substack has the following order:

```
:ROOT or :SEG
:INCLUDE   }
:EXCLUDE   }
:LIB       }
:LCOMMON   }   These commands may occur
:RES       }   in any order
:COMMON    }
Binary Object Module1  }
  .                    }
  .                    }  Binary object modules are in-
  .                    }  cluded at this point in the
Binary Object Modulen  }  substack only if the input
                       }  device specified on the
                       }  preceding :ROOT or :SEG
                       }  command is the same as the
                       }  "C" device.
:MODIFY
  .
  .
:MODIFY
```

The PUBLIB substack has the following order:

```
:PUBLIB
:INCLUDE   }   These commands may occur
:EXCLUDE   }   in any order.
Binary Object Module1  }
  .                    }
  .                    }  Binary object modules are
  .                    }  included in the substack
Binary Object Modulen  }  only if the input device
                       }  specified on the PUBLIB
                       }  command is the same as
                       }  the "C" device.
:MODIFY
  .
  .
:MODIFY
```

!OLOAD The ! OLOAD control command signifies that the Overlay Loader Processor is to be executed in the background area. Any error on the ! OLOAD control command causes the Loader to abort. Recovery consists of correcting the error and reloading the entire job.

If an ! OLOAD control command is continued to another card, the continuation command must have a colon (:) in column one instead of an exclamation (!) character.

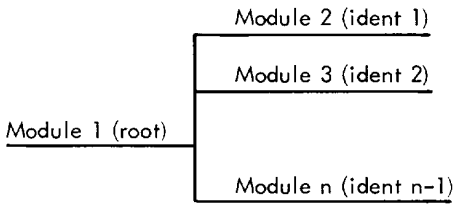
The form of the command is

```
! OLOAD [(option1)(,option2)...(,optionn)]
```

where the options are

GO specifies that the Loader is to input all object modules from the GO file and form a root. The only other control commands recognized in this mode are :RES, :INCLUDE, :MODIFY, and :ASSIGN. All other commands are considered illegal.

GO,LINKS specifies that the Loader is to form a link type overlay structure from GO in the following manner: module 1 is identified as the root (segment 0); module 2 is identified as segment 1 and is linked to the root, ...; module n is identified as segment n-1 and is linked to the root.



Libraries are searched at the end of each segment. Only :MODIFY and :ASSIGN commands are honored. The user must have explicit SEGLOAD calls to load segments 1, 2, ...n. No implicit calls are built by the Loader.

PUBLIB,name1[,name2,name3] specifies that the named Public Libraries are to be resident when the loaded program executes, and that the Loader is to establish the appropriate linkage. Name$_i$ is the file name of a Public Library in the Foreground Programs area of the RAD. The PUBLIB keyword may not be used when a Public Library is being created. (i.e., one Public Library cannot reference another Public Library).

LIB [,USER,SYSTEM] specify the Libraries to be searched following each segment. The order of the keywords USER,SYSTEM defines the order of the search. If the USER or SYSTEM keywords are omitted, and only the LIB keyword is specified, the Library search is suppressed. If the LIB option is omitted, the Loader searches the System Library after each segment.

Note: The :LIB control command overrides this option for the segment in which it appears (see below).

{FORE / BACK} [,fwa,lwa] specifies whether the program being loaded is to execute in foreground or background. If the option is omitted, the program will execute in the background area defined at SYSGEN. The "fwa" and "lwa" parameters are hexadecimal values denoting the first word address (on a doubleword boundary) and last word address of the area within which the program will execute. The PCB will be located at the FWA specified.

For foreground programs, the default "fwa" is the FWA of the foreground area (K:FGDBG2) and the default "lwa" is end of memory (K:FGDEND).

For background programs, the default "fwa" is the FWA of the background area (K:BACKBG), and the default "lwa" is the end of background defined at SYSGEN (K:FGDBG2-1). If the "fwa" specified for a foreground program lies in the background area, the background will be checkpointed when the foreground program is loaded. If the "fwa" and "lwa" specified for a background program exceed the limits of background at time of loading, the program is still loaded and may be executed by changing the upper limit of background. Note that "lwa" is an indicator of upper limit. If the program exceeds this limit, the user is warned but loading is not inhibited (except when a Public Library is being created). If the program loads in less space, the shorter area will be output in the header.

TASKS,value specifies the number of tasks in this program which are to be connected to interrupts. The option is used to allocate the INTTAB, and thus has meaning only for foreground programs. The default is 0 for Background and Public Libraries; 1 for Foreground. The "value" parameter is a decimal number.

TEMP,value specifies the number of words to allocate for the Temp Stack. The Temp Stack is located in part two of the root. The default size for a foreground or background program is 200 words. Public Libraries do not have Temp Stacks. Therefore, the option may not be specified when a Public Library is being created. The "value" parameter is a decimal number.

FILE,area,name specifies the RAD area and file name of the output file to which the loaded program is to be written (hereafter referred to as the Program File). The default assignment of the program file is OV in the Background Temp area. If the Background Temp area (BT) is specified, the file name must be OV. When a foreground program is being loaded, either the Foreground Programs area (FP), or Background Temp area (BT), must be specified. When a Public Library is being created, the Foreground Programs area (FP) must be specified.

MAP [,{PROGRAM / ALL}] specifies that a MAP of the program is to be output to M:LO. If no keyword follows MAP, a short map consisting of information about program allocation and overlays is output. If PROGRAM keyword is given, external definitions and control section designations for each segment are listed without library definitions. For the ALL keyword, both program and library definitions are listed. In default, no MAP is output. (Diagnostics and unsatisfied references are still listed on M:LL.)

BOUND,value sets the loading (and execution) origin for each object module to the next higher multiple of the bound value (e.g., if BOUND = 100, then an origin would change from 3EF to 400). The "value" parameter must be a hexadecimal number less than or equal to 1000 and a power of 2. Suggested values are 10, 100, or 1000. The BOUND does not apply to Library modules. If BOUND is not specified, the Loader begins each module on a doubleword boundary.

UDCB,value specifies the number of unnamed DCBs to be allocated by the Loader. (See "Loader-Generated Items" for details.) The "value" parameter is a decimal number.

STEP specifies a "WAIT" after loading each module from paper tape. Used in RBM ATTEND mode.

Example: Form Root From GO File Modules

```
!OLOAD GO,(TEMP,300),(MAP,PROG),(UDCB,3)
```

This example specifies that the Loader is to form the root from object modules located on the GO file, allocate 300 words for the Temp Stack, output a PROGRAM map, and allocate three unnamed DCBs.

:ROOT The ROOT control command is used to specify the object modules from which the root segment is to be created. The ROOT command must precede all SEG commands.

The form of the command is

```
:ROOT [(ENTRY,def) (,input option1),... (,input optionn)]
```

where

ENTRY,def specifies the location at which execution will commence after the root is loaded at execution time. The def parameter must be an external definition (1-8 EBCDIC characters) in the root segment. This entry point overrides all subsequent entry addresses encountered in loading. The default entry address is the last transfer address encountered in the nonlibrary object modules of the root.

Input options are of the form

$$\left\{\begin{array}{l}\text{DEVICE,type[,PACK]}\\ \text{FILE,area,name}\\ \text{OPLB,label}\end{array}\right\}\text{[,value]}$$

where

DEVICE,type specifies the input device in the format yyndd.

where

yy is a device type code.

n is the IOP to which the device is connected.

dd is the hardware device number of the device (e.g., CRA03,9TA81).

PACK specifies that the input is from 7-track magnetic tape with packed binary format.

FILE,area,name specifies the RAD area and name of the input file. If the Background Temp area (BT) is specified, the file name must be GO. Note that a file may be used as input to more than one segment (in different paths). A named file is rewound each time it is specified; the GO file is not.

OPLB,label specifies the operational label from which the object module(s) will be input. The "label" parameter must be a 2-character standard system operational label.

value either a decimal number ($1 \leq value \leq 8191$) that specifies the number of object modules to input from the specified device/file; or the text string, EOD, which means to input from the specified device/file until an !EOD is encountered. If value is omitted, one object module will be input from the specified device/file.

If there are no input options on the ROOT control command, one object module will be input from the GO file. Note that the order of the subfields determines the order in which the object modules are loaded.

Example: Form Root From Input File

```
:ROOT (FILE,D1,BETA,3),(ENTRY,START)
```

This example specifies that the root is to be formed from the three object modules in a file called BETA located in the D1 RAD area. After loading, execution is to commence at the location defined by the external definition START.

:SEG The SEG control command is used to define a segment's overlay linkage and to specify the object modules from which the segment is to be created.

The form of the command is

$$\text{:SEG (LINK,ident}_1\text{[,ONTO,ident}_2\text{])[,(EXLOC,addr),(ENTRY,def),(input option}_1\text{),...,(input option}_n\text{)]}$$

where

LINK,ident$_1$ specifies the identification number of the segment being loaded. The ident$_1$ must be specified and must be the same number used within the overlay program to call in the segment at execution time by SEGLOAD. The "ident$_1$" parameter must be a decimal number between 1 and 32,767.

ONTO,ident$_2$ specifies the identification number of the segment (which must have been previously loaded when this control command is interpreted) to which this segment is linked as an overlay. If ident$_2$ is absent, ident$_1$ is linked onto the root. The "ident$_2$" parameter must be a decimal number between 0 and 32,767 (0 denotes the root).

EXLOC,address specifies an optional execution address for loading of this segment. The "address" parameter is a hexadecimal value. This value will be bounded by either the specified or default BOUND. If the EXLOC option is omitted, the segment will be located at the first bounded address following the segment that this segment is linked to.

ENTRY,def specifies an entry point for the segment. The "def" parameter must be an external definition in an input module of the segment. The value of the "def" overrides any transfer addresses encountered in loading of the segment. The default entry address is the last transfer address encountered in loading nonlibrary ROMs.

The input operations are the same as for the ROOT control commands. If there are no input options on the SEG control command, a single object module from the GO file will be input.

The ROOT and SEG control commands must be input in an order determined by the overlay structure of the program. The segments in the example given in Figure 4 have been numbered to illustrate this order. Basically, segments are input one path at a time, with the restriction that segments common to more than one path are input only once.

Example 1.

The following control commands define the overlay structure of Figure 4. This example specifies that one object module for each segment will be input from the GO file.

```
:ROOT
:SEG (LINK,1,ONTO,0)
:SEG (LINK,2,ONTO,1)
:SEG (LINK,3,ONTO,1)
```

```
:SEG (LINK,4,ONTO,0)
:SEG (LINK,5,ONTO,0)
:SEG (LINK,6,ONTO,5)
:SEG (LINK,7,ONTO,6)
:SEG (LINK,8,ONTO,6)
:SEG (LINK,9,ONTO,5)
:SEG (LINK,10,ONTO,9)
:SEG (LINK,11,ONTO,10)
:SEG (LINK,12,ONTO,11)
:SEG (LINK,13,ONTO,11)
:SEG (LINK,14,ONTO,10)
:SEG (LINK,15,ONTO,9)
```

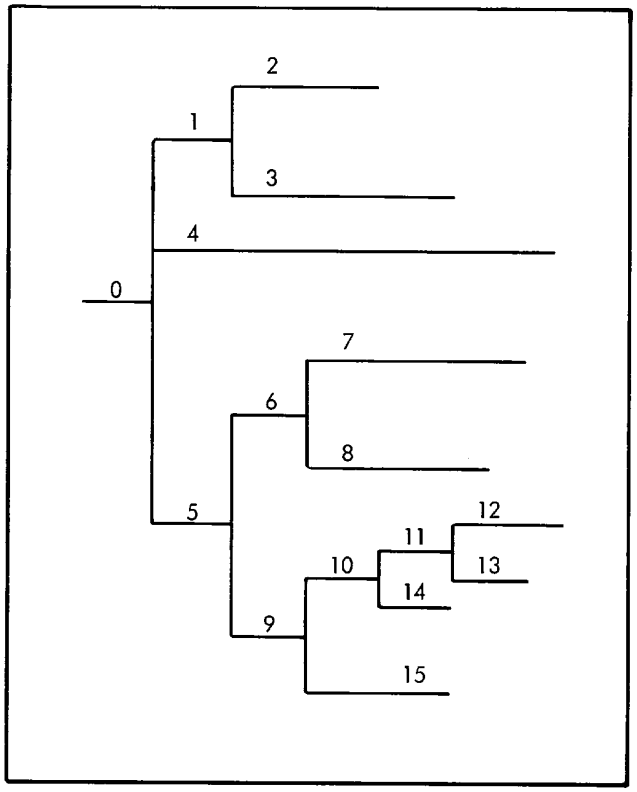


Figure 4. Overlay Example

Example 2.

The following control commands define the overlay structure illustrated in Figure 5. This example specifies that one object module for each segment will be input from the GO file.

```
:ROOT
:SEG (LINK,10,ONTO,0)
:SEG (LINK,5,ONTO,10)
:SEG (LINK,25,ONTO,10)
:SEG (LINK,103,ONTO,0)
```

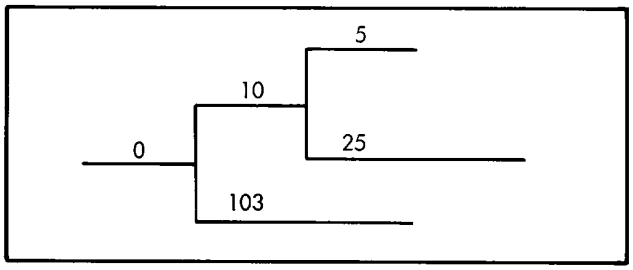


Figure 5. Object Module from GO File

BINARY OBJECT MODULES

The Loader inputs binary object modules from mixed media according to the input files and devices specified on the ROOT, SEG and PUBLIB commands. Files may be blocked or unblocked. Non-RAD input is written to a temporary RAD file for Pass 2. Binary modules are read sequentially from each RAD file. Each RAD file, with the exception of GO, is rewound each time that it is named as input on a control command. Therefore, multiple inputs from a file (other than GO) result in the file being reread from the beginning.

In this example,

```
:SEG (LINK,204,ONTO,0),(FILE,FP,PROG1,2);
:(FILE,BT,GO,4)
.
.
.
:SEG (LINK,205,ONTO,0),(FILE,FP,PROG1,5);
:(FILE,BT,GO,2)
```

the first access to the PROG1 file (in SEG 204) would result in the first two modules being loaded from the file. The second access (in SEG 205) would result in the first five modules of the file being loaded (not modules 3-7). The GO file is read contiguously throughout a pass, no matter how many accesses are made. In segment 204, the first four object modules from GO would be loaded. In segment 205, the next two modules (5 and 6) from GO would be loaded.

:LIB The LIB control command specifies the library search for one segment only (i.e., the segment identified by the preceding ROOT or SEG command). It overrides the library search specified by the LIB option on the OLOAD control command.

The form of the command is

```
:LIB [(USER,SYSTEM)]
```

where option keywords USER and SYSTEM are used to denote the libraries and order of search; e.g., :LIB (USER) would cause only the USER library to be searched for that segment. If neither USER nor SYSTEM is specified, library search (except for Public Library) is suppressed for that segment.

Example:

In Figure 6, assume an overlay program with four ROMs on the GO file; with segments 0, 1, and 2 coded in Macro-Symbol, and segment 3 coded in FORTRAN IV-H.

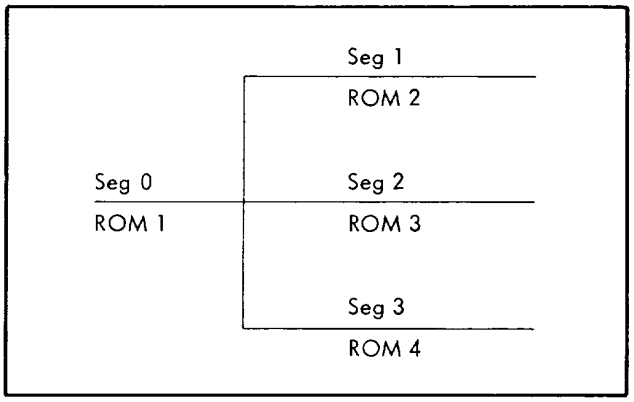


Figure 6. :LIB Command Usage

To speed up the load process, the :LIB command in the stack of commands given below would be used to specify a search of the System Library in segment 3, and the no-library search option would be specified on the !OLOAD command.

```
!OLOAD (MAP,ALL),LIB
:ROOT
:SEG (LINK,1)
:SEG (LINK,2)
:SEG (LINK,3)
:LIB SYSTEM
```

:INCLUDE The INCLUDE control command allows routines to be loaded from libraries when no reference to the routine has been made in any module of the segment.

The form of the command is

:INCLUDE (def_1 [,def_2,...,def_n])

where

def_i is the EBCDIC symbol of a definition contained in the library routine to be loaded. The symbol may be one to eight EBCDIC characters. The def_i must be available in a library specified in a preceding :LIB command or the LIB option on the !OLOAD command; any unfound def_i results in an error diagnostic.

Example. Load Two Routines From Library

```
:INCLUDE (9SETUP,7SET)
```

In this example, the routines 9SETUP and 7SET are to be included in the load from a library previously specified in the search criteria.

:EXCLUDE The EXCLUDE control command inhibits library search and linkage for the named definition(s) even though an external reference occurs in a module of the segment.

The form of the command is

:EXCLUDE (def_1 [,def_2,...,def_n])

where

def_i is the EBCDIC symbol of an external reference contained in a module of the segment. However, def_i must not occur as an external definition in a lower level segment of the path. The symbol may be one to eight EBCDIC characters. Note that EXCLUDE also inhibits linkage with the specified Public Library for the given symbols.

Example: Exclude Search For Named Routine

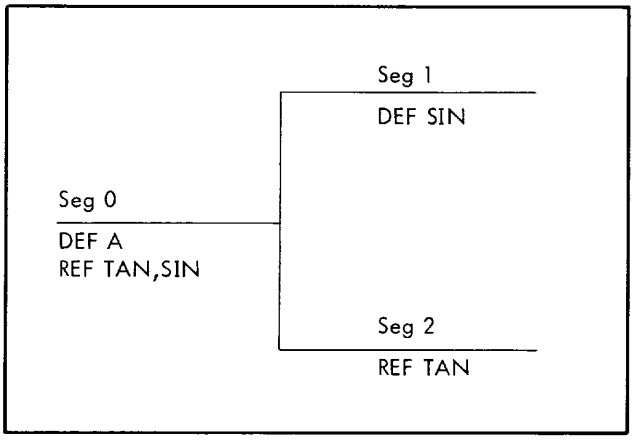


Figure 7. :EXCLUDE Command Usage

In this example, the tree structure illustrated in Figure 7 shows a routine called SIN in segment 1 that has the same name as a library routine, and is referenced in an earlier segment (Seg 0). The command

```
:EXCLUDE (SIN)
```

inhibits library search and linkage for the named routine only so that the TAN routine would be included in the root but the SIN routine would not.

:COMMON The COMMON control command specifies that the Loader is to set the base of Blank COMMON at the end of the segment identified by the preceding ROOT or SEG control command (see "FORTRAN Interface" for details). If this control command is not included, Blank COMMON is set at the end of the longest path. Only one COMMON control command may be used in a control command stack.

The form of the command is

```
:COMMON
```

The specification field must be blank.

:RES The RES control command allows the user to reserve and name one or more areas at the end of the segment for load-time or run-time debug purposes (see "MODIFY" control command for further comment).

The form of the command is

```
:RES (def,size)1 [,(def,size)2,...,(def,size)n]
```

where

def creates an external definition whose value is the FWA of the reserve area. The definition must be unique within the path.

size is a decimal value specifying the number of words in the reserve area.

Example: Reserve Two Areas at Segment End for Debug Purposes

```
:RES (PATCH1,5),(PATCH2,10)
```

In this example, two areas are being reserved at the end of a given segment; the first (PATCH1) comprising an area of five words and the second (PATCH2) an area of ten words.

:LCOMMON The LCOMMON control command allows the user to determine the allocation of labeled COMMON blocks (DSECTs) within the root and overlay segments of the program. (See "FORTRAN Interface-Labeled COMMON" for a discussion of restrictions concerning labeled COMMON allocation and initialization.)

The form of the command is

```
:LCOMMON (blockname,size)1 [,(blockname,size)2,
...,(blockname,size)n]
```

where

blockname is the one- to eight-character EBCDIC name of the labeled COMMON block or DSECT.

size is a decimal value specifying the largest word size needed for the allocated block. If 'size' is omitted, the first size encountered will be used.

Example: Specify Allocation of a Labeled COMMON Block

In the overlay structure given in Figure 8, seg 0 references a Labeled COMMON block A of 50 words; seg 1 references a Labeled COMMON block A of 100 words; seg 2 references a Labeled COMMON block A of 60 words.

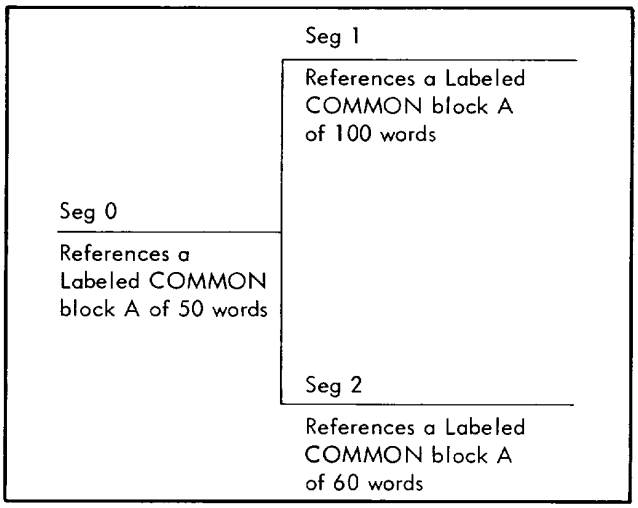


Figure 8. DSECT Allocation Example

Normally, the Loader would allocate block A with the size first encountered (50 words), and would output a diagnostic alarm when the second block of 100 words in seg 1 is encountered.

However, the :LCOMMON command inserted in the deck structure

```
!OLOAD (MAP,ALL)
:ROOT (DEV,CRA03)
:LCOM (A,100)
:SEG (LINK,1),(DEV,CRA03)
:SEG (LINK,2),(DEV,CRA03)
```

would set the size of block A to 100 words.

:MODIFY The MODIFY control command modifies core locations of relocatable programs at load time. Core locations in either root or overlay segments can be modified. Since the reserved area at the end of a segment (allocated with the RES command) is output to the RAD as part of the segment, that area can be used for "patches" that will be read in with the segment at execution time. The MODIFY commands must be input at the end of the ROOT, SEG, or PUBLIB substack for the segment being modified. If the GO option is specified, the MODIFY commands must follow any RES or INCLUDE commands and precede any :ASSIGN commands. If the (GO,LINKS) option is specified, the MODIFY commands must be ordered by segment number and follow the OLOAD command.

The form of the command is

```
:MODIFY [(SEG,ident),](LOC,address),word1,...[,wordn]
```

where

SEG,ident specifies the identification number of the segment to be modified. This option is only necessary when the (GO,LINKS) option has been specified. If this option is omitted, the segment identified by the preceding ROOT, SEG, or PUBLIB command will be modified.

LOC,address specifies the relative location of the first 32-bit word to be modified. The address must be expressed as an external definition name plus or minus an optional hexadecimal or decimal offset. Hexadecimal values are distinguished from decimal values by a period preceding the hexadecimal value (i.e., .A9B).

word$_i$ specifies the word to be inserted (right-justified) at the ith location. The word can be expressed as:

1. A signed (plus sign (+) optional) hexadecimal or decimal value that cannot be enclosed in parentheses. Hexadecimal values are preceded by a period.

   Examples

   -6, 100, .2A, -.AF

2. An external name plus or minus an optional offset that cannot be enclosed in parentheses. The offset can be either a hexadecimal or decimal value. Address resolution for the external can be specified by using the SYMBOL notation: rr(name) ± offset

   where

   rr = BA, HA, WA, or DA

   Word resolution is assumed by default. Note that BA(ALPHA)+3 is legal; BA(ALPHA+3) is not. If the name specified has not been declared an external somewhere in the overlay segment's path, it will be listed as an unsatisfied REF on the MAP.

   Examples:

   TABL + .F, TABL-1, TABL, HA(TABL)

3. A symbolic instruction that must be enclosed by parentheses. The mnemonic field of the instruction must be an EBCDIC operation code that immediately follows the left parenthesis. (The floating-arithmetic, floating-shift, decimal, and byte string instructions have not been implemented.) The register and index fields can only be signed hexadecimal or decimal values. The address field can be either a signed hexadecimal value, a signed decimal value, or an external name plus or minus an optional offset.

Examples:

```
:MODIFY  (LOC,MAP+. F0),(B PATCH+6)
```

```
:MODIFY  (LOC,PATCH+6),(LI,6 BA(TABL)),
(LW,9 *WA(VAL)+9,6),(B MAP+. F1)
```

```
:MODIFY  (SEG,0),(LOC,.140),.3F,-.F,250,-1
```

```
:MODIFY  (SEG,1),(LOC,CCI+.80),(LI,5 0),
(BAL,15 SERCHTAB),(MTW,-1 FLCHG),(BLEZ*CC1+5),
(LB,6 0,5),(STB,6 *WA(TABL)+1,5),(LW,0.4E)
```

```
:MODIFY  (SEG,3),(LOC,FPTLO),M:LO+.11000000,
.F0400090,ERRIO,ABNIO,BUF1,80,0
```

In reporting MODIFY command errors, any EBCDIC string, decimal number, or hexadecimal number that is separated by a comma, blank, plus sign, or minus sign (ignoring parentheses) is counted as an item. An example of items on a MODIFY command is given in Figure 9.

:ASSIGN The ASSIGN control command is used to create, initialize, or modify DCBs at load time. If the DCB

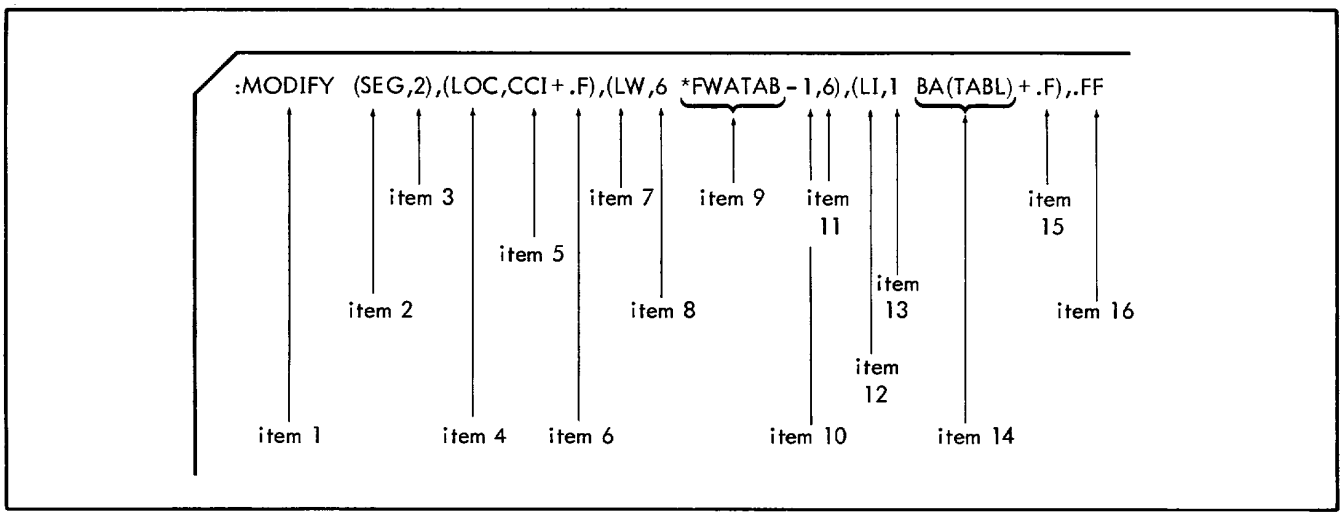


Figure 9. :MODIFY Command Items Example

is in the program's DCBTAB table, it will be either initialized or modified. If the DCB is not named in DCBTAB, the Loader will build the DCB from the parameters on the :ASSIGN control command in an unnamed DCB's entry. An error diagnostic is output if an unnamed DCB entry is not available (see "Load Time ASSIGN").

The format and options are identical to the Monitor !ASSIGN control command. The :ASSIGN control commands must be the last commands in the control command stack.

Example: Assign a DCB at Load Time

```
:ASSIGN (F:2,LPA02),VFC,(RECL,133)
```

In this example, the DCB F:2 is assigned to the line printer. Vertical format control is specified, so that the first byte in each record controls the spacing on the line printer. Although 133 bytes are output for each record, only the last 132 bytes are printed because VFC is specified.

:PUBLIB The PUBLIB control command is used to specify the object modules from which the Public Library is to be created. The order of the parameters determines the order of loading.

The form of the command is

```
:PUBLIB[(input option),(input option),...,(input option)]
```

where option may be

DEVICE,type[,PACK]
FILE,area,name
OPLB,label
} as for ROOT and SEG commands

If there are no input options on the PUBLIB control command, the first object module on the GO file will be input.

When the specified object modules have been input, the Loader searches the libraries (specified on the OLOAD control command or the System Library by default) to satisfy any unsatisfied primary references. If a COMMON, labeled COMMON block, or other DSECT is encountered in an object module of the Public Library, the load process is aborted and an error diagnostic is output. If the severity level exceeds zero in the load process, the Public Library is not loaded. If anything was written on the Public Library file, the file is destroyed and an error diagnostic is output.

The following conventions concerning other control commands should be observed when using the PUBLIB command:

1. The FORE option must be specified on !OLOAD to define the area that the Public Library is to occupy at execution time. If the limits of this area are exceeded, the Loader aborts.

2. The FILE option on !OLOAD must specify the name of the Public Library file being created in the Foreground area.

3. The TEMP, PUBLIB, GO, and TASKS options are illegal, and if used, the Loader will abort with an OLOAD control command error.

4. BOUND should be avoided unless a special debug version of a Public Library is being created.

5. ROOT, SEG, :ASSIGN, LIB, LCOMMON, RES, and COMMON control commands cannot be used in creating a Public Library.

Example: Create Public Library From Specified Module

```
:PUBLIB (FILE,SP,MODULE,10)
```

In this example, the Public Library will be created from the first 10 object modules in the System Library.

PROGRAM FILE

The Program File contains the root and overlay segments in core image format and a one-granule header. The program header is located at granule 0 and contains information necessary to run-load the program.

ROOT SEGMENT

The root is divided into two parts (see "Core Layout at Execution Time" later in this chapter). Part one of the root always begins in granule 1 of the Program File, and contains the PCB, root code, library code, labeled COMMON, and RES area for the root. Part two contains the DCBTAB, INTTAB, OVLOAD Table, Loader-created DCBs, and the Temp Stack.

The Temp Stack is not output on the Program File. Each part of the root is written as a continuous string of bytes. There is no restriction on the size of the root.

OVERLAY SEGMENTS

Each overlay segment begins on a granule boundary and is written on the Program File as a continuous string of bytes. The order of segments on the file is unimportant, since the granule displacement pointer (in the OVLOAD table) for each segment specifically determines its position. Segments cannot be longer than 16K words (64K bytes).

TEMPORARY RAD FILES

The Loader uses six scratch files in the Background Temp area of the RAD (X1,X2,...,X6). If one of these files overflows, the Loader completes the pass over the object modules even though the load will be aborted. The Loader calculates the number of records (for sequential files) or granules (for direct-access files) required for all scratch files and lists this information on the Map. With this information, the user can then allocate the Background Temp files with an !ALLOBT command and reload the program.

LOADER-GENERATED ITEMS

All items discussed in the following paragraphs are generated by the Loader and located in the root segment of the overlay program (see Figure 9 in this chapter for a diagram of the core allocation).

PROGRAM CONTROL BLOCK

The PCB is built by the Loader and located at the FWA of the overlay program area.

DATA CONTROL BLOCKS

The Loader automatically includes a copy of the M:SL DCB in any program that has overlay segments. (M:SL is used by the Monitor SEGLOAD function to read in overlay segments at execution time.)

Any external DEF/REF that begins with an M: or F: is defined to be either a system (M:) DCB or user (F:) DCB. DCBs referenced by the program that are not satisfied at the conclusion of the load process are either created or allocated by the Loader. Copies of system and FORTRAN DCBs are created with their standard system parameters and operational label assignments. Space for user DCBs is allocated at the rate of seven words per DCB.

Parameters for user or system DCBs may be defined by either !ASSIGN control commands at execution time (for background programs only) or :ASSIGN control commands at load time.

The user can create his own DCBs within the source code and locate them in any segment of his overlay program. However, if the user wishes to change parameters in a DCB at execution time via an !ASSIGN command, he must declare the DCB as an external definition (with a name that begins with an F:) and locate the DCB in the root segment. To utilize the FORTRAN IV-H capability of performing I/O by using variables as operational labels, the user can specify (on the OLOAD control command) a number of unnamed DCBs to be allocated by the Loader. The user must name and define these DCBs before the program executes; either at load time (with :ASSIGN), or execution time (with !ASSIGN).

DCBTAB

The DCBTAB table is created by the Loader, and contains the EBCDIC name (if any) and absolute core location of each Loader-recognized or created DCB in the root of the overlay program. The EBCDIC name of an unnamed DCB is inserted when the DCB is given a name by either the !ASSIGN or :ASSIGN control command.

INTTAB

The interrupt task table (INTTAB) has a one-byte entry for each interrupt task in a foreground program. Space for the INTTAB is allocated by the Loader according to the number of tasks specified on the OLOAD control command.

OVLOAD TABLE

The OVLOAD table is built by the Loader and contains the information necessary for SEGLOAD to read in overlay segments at execution time. The OVLOAD table consists of one entry for each overlay segment, with a total of eleven words per entry.

TEMP STACK

The Loader allocates space for the overlay program's Temp Stack either according to the number of words specified on the OLOAD control command, or by default. The Temp Stack is located at the end of part two of the root segment and is not output on the Program File (see "Core Layout of User Program at Execution" later in this chapter).

EXTERNAL DEFINITIONS

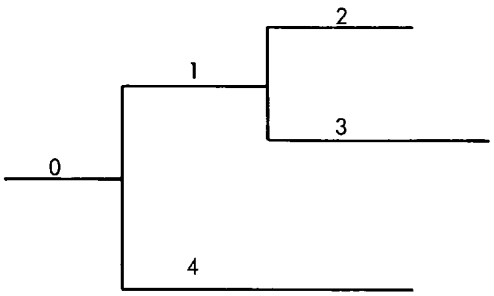
The Loader adds the external DEFs F4:COM and P:END to all programs except for Public Libraries. F4:COM is the name of FORTRAN's blank COMMON. The initial size is set to zero and changed to the largest size encountered during the load process. If there are no references to F4:COM, blank COMMON is allocated with a size of zero. The Loader indicates the LWA+1 (including blank COMMON) of the loaded overlay program by an external definition, P:END. External references to P:END within the overlay program will be linked to this definition.

The external DEF, FP:MBOX, is added to Foreground overlay programs by the Loader only if an area was allocated at SYSGEN time. FP:MBOX is the name of the Foreground Program's Mailbox. External references to FP:MBOX will be linked to this definition.

LIBRARIES

The Overlay Loader supplies the capability to search the System Library or the User Library in any order. The default condition is for the Loader to search and load only from the System Library. Control commands and keywords enable the user to control more specifically the search and load options. Note that an attempt will first be made to satisfy all REFs with DEFs from the Public Library, if a Public Library has been specified on the OLOAD control command.

If any unsatisfied primary references exist after loading the specified modules for a root or an overlay segment, the Loader searches the library or libraries in the specified order to satisfy those references. Thus, if an external REF is made to a higher level segment, the name should not be the same as a library definition. Consider the following:



If segment 1 contains a primary reference, 9SIN, it will normally be satisfied by loading a Library at the completion of segment 1. Thus, if the definition 9SIN occurred in segment 2, it would be in error (a duplicate definition). The loading of 9SIN from the library can be suppressed by using the EXCLUDE command. In this case, the forward REF would be linked and no duplicate DEF would occur. However, if the definition 9SIN occurred in the root, or in the library loaded in the root, no search for 9SIN would be made in segment 1, and the occurrence of the definition 9SIN in segment 2 would be in error. Primary references can occur in two ways: as external references in a module, or by listing the primary references on the INCLUDE control command.

SYSTEM AND USER LIBRARIES

Cross-references between System and User Libraries are allowed. However, since each library is searched only once per segment, the order of search is important.

If Library A contains references to be resolved by Library B, the search criteria :LIB (A,B) must be specified to guarantee cross-reference resolution. If B also contained references to A they would not be resolved. (Note that these remarks do not apply to cross-references within any single library).

Generally, the System Library should contain the FORTRAN Math and Run-Time Routines and should be independent. The User Library is a repository for user subroutines and alternate Math and Run-Time routines that supersede the same routines in the System Library.

The typical search order would be

```
:LIB (USER,SYSTEM)
```

where both libraries are referenced. In this case, all unsatisfied REFs from the User Library would be satisfied (where possible) from the System Library.

ASSEMBLY LANGUAGE

Library routines may be coded in Symbol, Macro-Symbol, FORTRAN IV, or FORTRAN IV-H.

ENTRY ADDRESS

Entry addresses in library routines are ignored.

SYSTEM AND USER LIBRARIES ON RAD

The System Library and the User Library on the RAD are structurally identical. Each library consists of four files:

EBCDIC
MODIR
DEFREF
MODULE

The System Library is located in the System Programs (SP) area of the RAD. The User Library is located in the Foreground Programs (FP) area of the RAD.

Only the MODULE file contains the actual binary modules of the library. The other files are tables constructed by the RAD Editor to facilitate the rapid search of the library by the Overlay Loader without actually reading the module. The library is structured on the principle that access should be as fast as possible, since it is performed frequently during an overlay loading procedure.

The three files: EBCDIC, MODIR, and DEFREF contain enough information to determine which modules from the actual MODULE File are to be loaded without examining these modules directly. All four library files are constructed and maintained by the RAD Editor. These short files contain coded information about the external definitions and primary references for each module in the library.

CONSTRUCTING AND MAINTAINING LIBRARY

To begin construction of a library, the user allocates the EBCDIC, DEFREF, MODIR, and MODULE files with the RAD Editor, and then copies the library's binary object modules onto the MODULE file. As each module is copied, the DEFs and REFs are scanned, and corresponding entries are built in the other files by the RAD Editor. Library routines may be added or deleted by using the RAD Editor :COPY and :DELETE commands.

PUBLIC LIBRARY

The Public Library is a file containing a set of reentrant subroutines in core image format that can be shared in common by all foreground and background programs. The resultant saving in core can be considerable where a FORTRAN library is shared. The Public Library is created from input modules or library routines by the Loader (see "Forming a Public Library"). The availability of the Public Library is determined at execution time.

CALLING THE PUBLIC LIBRARY

When a user indicates by the PUBLIB keyword on the OLOAD control command that Public Libraries are to be used to satisfy references, the names are set in the program header for the Root Loader, and the Public Library Symbol tables are read from the Public Library files and added to the loaded program's Symbol table. The Loader will satisfy primary external references with Public Library definitions at the time the external reference is encountered in the object module, not at the end of the segment (as when the other libraries are searched). When the Root Loader loads the root segment of a program, the header is searched to determine if the program contains the name of one or more Public Libraries. If so, and one of the named Public Libraries is not already in core, the Monitor determines whether Public Library space is available. If available, the Root Loader reads in the named Public Library or Libraries and the program executes. If the space is not available for all Public Libraries referenced, the program will be neither root loaded nor executed.

Each Public Library file is designated at Public Library creation time (see "Forming a Public Library"). All Public Libraries are located in the Foreground Programs area of the RAD.

LIBRARY PROTECTION

Since the call to a Public Library routine is by a BRANCH AND LINK(BAL) operation, the write key of the library routine is the same as the write key of the user program. Thus, the foreground and Monitor are both protected from being destroyed by background use of the Public Library. However, because of the write-lock protection, routines containing their own local storage (e.g., FORTRAN I/O run-time) may not be included in a Public Library that is to be called from the background since any attempt by a Public Library routine to write in its own foreground local storage with a background write key would cause a write-lock protection violation.

RELEASING A PUBLIC LIBRARY

If no currently executing program is utilizing a Public Library and the space is required to load a foreground program, the space is released.

FORMING A PUBLIC LIBRARY

A Public Library is created by using the :PUBLIB control command in place of the ROOT and SEG commands, and modules may be input and libraries searched and loaded in the same manner as for standard loading. Because each Public Library has a unique name, more than one Public Library can exist in the system. Although no more than three Public Libraries can be called by any one program, any number can be created.

ROUTINES USED TO FORM A PUBLIC LIBRARY

All routines used to form a Public Library must be reentrant. If the Public Library is to be used by background programs, all the routines must use the Temp Stack directly for local storage (e.g., FORTRAN IV Math Library; see "Protection", discussed previously).

If the Public Library is to be used only by the foreground, the method of moving local storage in routines to the Temp Stack on reentrance can be employed (e.g., Real-Time FORTRAN subroutines and FORTRAN Run-Time Library). FORTRAN main routines are not reentrant and cannot be used.

Routines assembled in Symbol (one-pass), or Macro-Symbol (two-pass), are acceptable provided the reentrancy requirements are met.

No references to COMMON, labeled COMMON, or DSECTs are allowed in any Public Library routine.

Since DCBs in the Public Library could not be assigned and might not be reentrant, DCBs will not be allowed in any Public Library routine. Note that it is not possible for the Loader to warn the user about DCBs that are not named according to the conventions and made externals.

The file associated with each Public Library is in the FP area. This file contains the actual core image of the Public Library and the corresponding Symbol table used by the Loader. The name of the file must correspond to the name given with the FILE keyword on the OLOAD control command, and the file must be previously allocated in the FP area by the user. If loading of the requested modules and libraries has been completed and there are no remaining unsatisfied primary references, the Loader writes the core image and the Symbol table to the file in the FP area. If unsatisfied primary references are found, the file in the FP area is destroyed. A file name of a previous Public Library may be used, but at the risk of obliterating the old file if the new one cannot be completed.

MAP

Three types of maps may be output to M:LO following Pass 2 according to the MAP keyword on the !OLOAD control command: a SHORT map, PROGRAM map, or ALL map. If the MAP option is not specified, none is output.

The short map is output when the MAP keyword appears alone. It consists of essential information about the overlay structure.

The PROGRAM Map consists of all elements of the short Map, plus all external definitions and control sections contained in the input modules (excluding those from Library ROMs).

The ALL Map consists of all elements of the PROGRAM map and includes all definitions and control sections from Library ROMs. A typical PROGRAM map is illustrated in Figure 10.

In Figure 10, the header keywords have the following meaning:

1. Program Header Keywords:

   FILE: Area and name of the program file.

   NUMBER OF OVERLAY SEGMENTS: Decimal number, excluding root.

   LIMITS: FWA and LWA of the Program area.

   BOUND: Hexadecimal value on which object module addresses are bounded.

   BLANK COMMON BASE: FWA of blank COMMON with the SIZE specified in decimal words.

   PUBLIC LIBRARIES: Names of the Public Libraries, if any, referenced by the program.

   TOTAL FILE SIZE: Number of hexadecimal/decimal WORDS output to the program file and the number of hexadecimal/decimal GRANULES required for the file.

   LIBRARY SIZE: Total number of words loaded from the user and/or system libraries.

   PROGRAM ERROR SEVERITY: Set to one if any kind of error is encountered; otherwise, set to zero.

2. Root Header Keywords:

   INPUT: Total number of hexadecimal words in the root loaded from the ROMs (excluding the Temp Stack).

   LIBRARY: Total number of hexadecimal words in the root loaded from the User Library and/or System Library.

   TOTAL SIZE: Total number of hexadecimal/decimal words in the root (including the Temp Stack).

   P1FWA: FWA of part one of the root.

   P1LWA: LWA + 1 of part one of the root.

   SIZE: Number of hexadecimal words in part one of the root.

   P2FWA: FWA of part two of the root.

   P2LWA: LWA + 1 of part two of the root.

   SIZE: Number of hexadecimal words in part two of the root.

   OVLOAD: FWA of the OVLOAD table.

   PCB: FWA of the Program Control block (PCB).

   ENTRY: Entry address for the root.

   TSFWA: FWA of the Temp Stack.

   SIZE: Number of decimal words in the root.

   DCBTAB: FWA of the DCBTAB.

   INTTAB: FWA of the INTTAB on zero, if none.

   INSEV: Set to one if the error severity level is set on any ROM input for the root; otherwise, it is set to zero.

   LDSEV: Set to one if any loading errors were encountered on the root; otherwise, it is set to zero.

3. Segment Header Keyword:

   INPUT: Total number of hexadecimal words in the segment loaded from the ROMs, RES, and LCOMMON control commands.

   LIBRARY: Total number of hexadecimal words in the segment loaded from User and/or System Libraries.

   TOTAL SIZE: Total number of hexadecimal/decimal words in the segment.

   FWA: FWA of the segment.

   LWA: LWA + 1 of the segment.

   ENTRY: Entry address of the segment.

   INSEV: Set to one if the error severity level is set on any ROM input for the segment; otherwise, it is set to zero.

   LDSEV: Set to one if any loading errors were encountered for the segment; otherwise, it is set to zero.

   GRAN: The granule number (decimal) on the program file where the segment's core image begins.

4. Control Sections:

Control sections input from the program ROMs are listed with the following information:

ROM	ROM number	address (hex.)	size (dec.)

Control sections input from user and/or system libraries are listed with the following information:

{ULIB / SLIB}	Record displacement in the MODULE file	address (hex.)	size (dec.)

```
PROGRAM MAP
BACKGROUND PROGRAM


 FILE  BT,OV
 NUMBER OF OVERLAY SEGMENTS       2
 LIMITS  FWA   2600   LWA   3032
 BOUND    100
 BLANK COMMON  BASE   2F96  SIZE       0
 PUBLIC LIBRARIES   NONE
 TOTAL FILE SIZE    AD2/  2770  WORDS   22/  34  GRANULES
 LIBRARY SIZE    489/  1161  WORDS
 PROGRAM ERROR SEVERITY  1


ROOT


 INPUT LIBRARY  TOTAL SIZE  P1FWA  P1LWA   SIZE  P2FWA  P2LWA   SIZE
   27A    3A4    689  1721   2600   2C1E    61E   2F96   3031     9B


 OVLOAD   PCB  ENTRY  TSFWA   SIZE  DCBTAB INTTAB  INSEV  LDSEV
  2F96  2600   2/08   2FE6     75   2FDO      0     1      1

  CONTROL SECTIONS
    ROM       1       2700       309
   ULIB     155       2836        12
   ULIB     156       2842        22
   ULIB     158       2858        16
   ULIB     160       2868        51
   SLIB       8       289C         5
   SLIB     203       28A2         6
   SLIB     204       28A8        15
   SLIB     205       28B8        23
   SLIB     207       28DO         6
   SLIB     208       28D6        18
   SLIB     210       28E8        24
   SLIB     379       2900        37
   SLIB     395       2926         2
   SLIB     396       2928         7
   SLIB     4C8       2930       118
   SLIB     637       29A6       186
   SLIB     653       2A60        23
   SLIB     656       2A78        56
   SLIB     659       2ABO       166
   SLIB     677       2B56        21
   SLIB     679       2B6C       109

     UDCB          F:S0          2700
     SDCB          M:L0          2FAD
     SDCB          F:3           2FB4
     SDCB          M:R0          2FBB
     SDCB          M:SL          2FC2
   U SDCB          UNNAMED       2FC9

     DSCT   IM     F4:COM        2F96  0          0
      DEF   IM     P:END         3031  0
      DEF   SL     SQRT          289C  0
   U  REF   IM     UNSAT
      DEF   UL     GAUSS         2868  0
     DSCT   IM     COUNT         2BDA  0         50
     DSCT   IM     CALC          2C0C  0         17
     DSCT   IM     MATRIXA       260E  0         27
      DEF   IM     CODE         FFFFD  C
      DEF   IM     MAXTAB        2813  C
      DEF   IM     MAIN          2708  0
     SREF   IM     7SET          2F71  0
```

Figure 10. Sample PROGRAM Map

```
      DEF  UL    SDELAY          2836  0
      DEF  UL    DELAY           2842  0
      DEF  UL    DERIVT          2858  0
  U   REF  UL    FUN1
      DEF  SL    9SETUP1         2928  0
      DEF  SL    9SQRT           290B  0
      DEF  SL    9CADD           28A2  0
SEGMENT    101  LINKED TO        0

 INPUT LIBRARY  TOTAL SIZE    FWA    LWA   ENTRY  INSEV  LDSEV  GRAN
   1B1     E5    296    662   2D00   2F96   2E00    0      0      19

  CONTROL SECTIONS
    ROM       1       2E00      153
   SLIB       6       2E9A       11
   SLIB      14       2EA6        5
   SLIB      16       2EAC        5
   SLIB     302       2EB2       29
   SLIB     310       2ED0       70
   SLIB     367       2F16       90
   SLIB     399       2F70       14

      DEF  SL    TAN             2EAC  0
      DEF  SL    ASIN            2EA6  0
      DEF  SL    ATAN            2E9A  0
     DSCT  IM    LOADS           2F7E  0        23
     DSCT  IM    MATRIXC         2D00  0        15
      DEF  IM    SEGONE          2E00  0
      DEF  SL    9ATAN1          2F27  0
      DEF  SL    9ATAN2          2F28  0
      DEF  SL    9SETUPM         2F70  0
      DEF  SL    9ASIN           2EB7  0
      DEF  SL    9TAN            2EE2  0
      DEF  SL    9ACOS           2EB6  0
      DEF  SL    9SETUPN         2F70  0
      DEF  SL    7SET            2F71  0
SEGMENT    102  LINKED TO        0

 INPUT LIBRARY  TOTAL SIZE    FWA    LWA   ENTRY  INSEV  LDSEV  GRAN
   1CE      0    1CE    462   2D00   2ECE   2D00    0      0      27

  CONTROL SECTIONS
    ROM       1       2D00      411

     DSCT  IM    TOTALS          2E9C  0        50
      DEF  IM    SEGTWO          2D00  0
LOADING WAS COMPLETED

  FILE  BT,0V          USED    34  GRANULES

  FILE  BT,X1          USED     0  GRANULES

  FILE  BT,X2          USED     4  GRANULES

  FILE  BT,X3          USED     4  GRANULES

  FILE  BT,X4          USED     4  GRANULES

  FILE  BT,X5          USED     4  GRANULES

  FILE  BT,X6          USED     0  GRANULES

WARNING: UNSATISFIED REF'S


END OF MAP
TOTAL JOB TIME=00:01:00
```

Figure 10. Sample PROGRAM Map (cont.)

5. DCBs

The user and system DCBs are listed after the control section of the ROOT with the following information:

{SDCB / UDCB} {name / UNNAMED} address (hex.)

DEFs, REFs, and DSECTs

The externals are listed with the following information:

a. {U / D} = unsatisfied or undefined / = doubly defined or referenced

b. {DSCT / DEF / REF / SREF} = dummy section / = definition / = primary reference / = secondary reference

c. {PL / UL / SL / IM} = Public Library / = User Library / = System Library / = Input Module

d. The symbol name in EBCDIC (one to eight characters).

e. If the definition is an address, it is expressed as a word address and a byte offset. If the definition is a constant, it is expressed as a hexadecimal number followed by the letter 'c'. For undefined symbols, the address field is blank.

f. The DSECT size in words (decimal).

ERROR DIAGNOSTICS

The Overlay Loader outputs diagnostic messages to M:OC and M:LL. Duplication is suppressed if OC and LL are assigned to the same device.

If an operator response is required, the Loader will call the Monitor "WAIT" function. The operator should hit the console interrupt and key in one of the following:

C	Continue
X	Abort
COC	Read the corrected command from M:OC and continue (used only in response to control command errors).

Note that the Monitor "WAIT" routine aborts if an !ATTEND control command has not been encountered in the job stack. The diagnostic messages in Table 14 are output by the Overlay Loader.

Table 14. Overlay Loader Diagnostics

Text	Meaning	Action
BACKGROUND TOO SMALL	User's program cannot be loaded in the current size of the background. This is a function of the number of external symbols and forward references that a program has, not a function of the program length.	Abort
BINARY CARD ENCOUNTERED INSTEAD OF CC	A binary record was encountered on the C device instead of a control command	Wait
BOT ON {yyndd / area,name}	Unexpected beginning-of-tape has been encountered on the specified device/file.	Abort
BUF SMALLER THAN DATA RECORD DCB x:xxxxxx	Specified DCB has been assigned to a record size larger than the I/O buffer associated with the Read request. Either the user has assigned incorrectly or the Loader has a program error.	Abort
CC ERR: DUP NAME IN ITEM xx	Item number xx on the command is a duplicate of a name in the symbol table.	Wait

Table 14. Overlay Loader Diagnostics (cont.)

Text	Meaning	Action
CC ERR: DUP SEG IDENT	Ident on :SEG command is a duplicate of a previous segment's ident.	Wait
CC ERR: FOLLOWING ITEM xx	There is an error following item xx on the command (e.g., a parameter has been omitted, an extra parameter has been included, etc.).	Abort if !OLOAD CC. Wait if any other CC.
CC ERR: ILLEGAL CC SEQUENCE	Loader commands have been ordered incorrectly.	Wait
CC ERR: ILLEGAL OPTION FOR PUBLIB LOAD (PUBL,name)	Option (PUBLIB,name) was specified on !OLOAD and a :PUBLIB command has been encountered.	Abort
CC ERR: ILLEGAL OPTION FOR PUBLIB LOAD (TASKS,value)	Option (TASKS,value) was specified on !OLOAD and a :PUBLIB command has been encountered.	Abort
CC ERR: ILLEGAL OPTION FOR PUBLIB LOAD (TEMP,value)	Option (TEMP,value) was specified on !OLOAD and a :PUBLIB command has been encountered.	Abort
CC ERR: ILL OPCODE IN ITEM xx	Specified operation code is either illegal or unimplemented.	Wait
CC ERR: ILL SEG IDENT	Seg ident on the :MODIFY command does not match the segment being modified.	Wait
CC ERR: ITEM xx	Item number xx on the command is in error.	Abort if !OLOAD CC. Wait if any other CC.
CC ERR: MODIFY OUTSIDE SEG LIMITS	One of the locations on the :MODIFY command is outside the limits of the segment.	Wait
CC ERR: NEED (FORE,fwa,lwa) OPTION FOR PUBLIB LOAD	Option (FORE,fwa,lwa) was not specified on the !OLOAD command and a :PUBLIB command has been encountered.	Abort
CC ERR: SEG IDENT NOT 1ST OPTION	Segment ident (LINK,$ident_1$) is not the first option on the :SEG command.	Wait
CC ERR: SEGMENTS ORDERED INCORRECTLY	:SEG commands have been input in the wrong order.	Wait
CC ERR: SPECIFIED AREA FOR PUBLIB LOAD NOT 'FP'	Option (FILE,area,name) on !OLOAD did not specify the Foreground Programs area (FP) and a :PUBLIB command has been encountered.	Abort

Table 14. Overlay Loader Diagnostics (cont.)

Text	Meaning	Action
CC ERR: STEP OPTION ILLEGAL WITHOUT ATTEND	An !ATTEND command must be included in the job when the STEP option is specified.	Abort
CC ERR: UNDEFINED FILE area,name	Area and file specified in the option (FILE,area,name) has not been defined by the RAD Editor	Abort if !OLOAD CC. Wait if any other CC.
CC ERR: UNDEFINED SYMBOL IN ITEM xx	Symbol name in item xx on the :MODIFY command has not been defined.	Wait
DCB CANNOT BE A DSECT SEGxxxxx {ULIB / ROM / SLIB} xxx	A DCB was encountered in the named segment that was assembled as a dummy section instead of being part of the control section.	Abort
DCB HAS BAD PARAMETERS DCB x:xxxxxx	Specified DCB has bad parameters. Either the user has assigned incorrectly or the Overlay Loader has a program error.	Abort
DCB HAS INSUFFICIENT INFO DCB x:xxxxxx	Specified DCB contains insufficient information to open a Read or Write operation. Either the user has assigned incorrectly or the Loader has a program error.	Abort
DCB NOT ASSIGNED DCB x:xxxxxx	Specified DCB has been assigned to the "null" device. Either the user has assigned incorrectly, or the Overlay Loader has a program error.	Abort
DEFAULT ENTRY ADDRxxxxx SUPPLIED FOR ROOT	A transfer address was not encountered on any ROM in the root and an entry address was not specified on the CC; therefore, a default has been supplied.	Continue
DSECT'S IN PUBLIB LOAD SEGxxxxx {ULIB / ROM / SLIB} xxx	Labeled COMMON blocks (DSECTs) are illegal in the Public Libraries.	Abort
EOT ON {yyndd / area,name}	End-of-tape has been encountered on the specified device/file.	Wait
EXLOC TOO LARGE SEGxxxxx	The execution locations of the specified segment will exceed 131K at the given EXLOC.	Abort

Table 14. Overlay Loader Diagnostics (cont.)

Text	Meaning	Action
FILE DESTROYED area,name	Overlay Loader is aborting past the point where data has been written on the specified program file. The first sector of the file has been zeroed out.	Abort
FILE UNCHANGED area,name	Overlay Loader is aborting at a point where the program file is unchanged.	Abort
ILLEGAL LOAD LOCATION xxxxx SEGxxxxx {ULIB / ROM / SLIB} xxx	Specified "load location" origin has been defined with a value that is either not an address or that lies outside the address limits of the specified segment. (A labeled COMMON block must be initialized in the segment where the block is allocated.)	Abort
ILL SEG IDENT TERMINATED MODIFY'S	The :MODIFY commands have been ordered incorrectly for the (GO,LINKS) option. The MODIFY mode has been terminated at this point. If the user wishes to continue, all :MODIFY commands that follow will be ignored.	Wait
LIB ROM'S EXCEED MAX SEGxxxxx	Maximum number of library ROMs that can be loaded is 2000.	Abort
MONITOR CC ENCOUNTERED INSTEAD OF :ROOT or :PUBLIB	Monitor control command was encountered on the C device instead of a :ROOT or :PUBLIB command.	Abort
MOUNT PAPER TAPE ROM	STEP option was specified on !OLOAD and the next relocatable object module (ROM) is to be input from the paper tape reader.	Wait. Operator should load the paper tape, interrupt, and key in "C".
PROGRAM ERR: {CCI / ONE / TWO / MAP / LIB} ADDR xxxx SEGxxxxx {ULIB / ROM / SLIB} xxx	Loader has a program error in the named overlay at the specified address.	Abort. Operator should get a core dump.
PROGRAM ERR: {CCI / ONE / TWO / MAP / LIB} SB=xx,ADDR xxxx DCB x:xxxxxx	Specified error status has been returned from an Overlay Loader call (in the named overlay) to a Monitor I/O routine. The address of the CAL and the name of the DCB are specified.	Abort

Table 14. Overlay Loader Diagnostics (cont.)

Text	Meaning	Action
PROGRAM ERR: UNALLOCATED CSECT SEGxxxxx {ULIB / ROM / SLIB} xxx	Loader has encountered a control section that has not been allocated; either a Loader, compiler, or assembler error.	Abort
RAD FILE TABLE FULL	RAD File Table size that was allocated at SYSGEN is insufficient.	Abort
READING AN OUTPUT DEVICE DCB x:xxxxxx	A DCB that the Overlay Loader reads with has been assigned to an OUT device. Either the user has assigned incorrectly or the Loader has a program error.	Abort
ROM ERR: BAD SEQ SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	Sequence number of the binary record does not equal xxx.	Wait
ROM ERR: CHKSUM SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	Specified binary record has a checksum error.	Wait
ROM ERR: EXPRESSION SIZE EXCEEDS MAX SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	An object language expression on the specified binary record exceeds 120 bytes.	Abort
ROM ERR: ILLEGAL LOAD ITEM SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	Object language on specified binary record cannot be translated (assembler or compiler error).	Abort
ROM ERR: NO MODULE END SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	Module end was not encountered on the last binary record of the relocatable object module.	Abort
ROM ERR: NOT OBJECT LANGUAGE SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	Specified binary record is not in object language format.	Wait
ROM ERR: NOT STANDARD BIN SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	Specified record has a non-standard binary format.	Wait
UNDEFINED FILE area,name DCB x:xxxxxx	Specified DCB has been assigned to a RAD file that has not been defined by the RAD Editor.	Abort

Table 14. Overlay Loader Diagnostics (cont.)

Text	Meaning	Action
UNDEFINED ORIGIN SEGxxxxx {ULIB / ROM / SLIB} xxx	Loader has encountered a "load location" origin with an expression that cannot be resolved.	Abort
UNEXPECTED EOD ON {yyndd / area,name}	Unexpected !EOD was encountered on the specified device/file.	Wait if the !EOD was encountered instead of a ROM; otherwise, Abort.
UNEXPECTED MONITOR CC ON {yyndd / area,name}	Unexpected Monitor control command was encountered while reading ROMs from the C device.	Abort
UNRECOVERABLE RD ERR ON {yyndd / area,name}	Transmission error has occurred while reading from the specified device/file.	Abort
UNRECOVERABLE WR ERR ON {yyndd / area,name}	Transmission error has occurred while writing on the specified device/file.	Abort
WARNING: DCB IN OVERLAY SEGMENT SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx DCB x:xxxxxx	Specified DCB was declared an external DEF in a segment other than the Root. The DCB will not be included in DCBTAB.	Continue
WARNING: DEF'D DCB NOT DEFINED DCB x:xxxxxx	Specified DCB was declared on external DEF and the DEF was never defined.	Continue
WARNING: DUPLICATE DEF'S	User's program contains duplicate external DEFs. Map will indicate the name(s) of the DEFs.	Continue
WARNING: DUPLICATE REF'S	User's program contains duplicate external REFs. Map will indicate the name(s) of the REFs. Occurs when identical DEFs in different segments of different paths are referenced by the same REF (in a segment common to both paths).	Continue
WARNING: ENTRY ADDRxxxxx OUTSIDE SEGMENT SEGxxxxx	Entry address for the specified segment is outside the segment's address limits.	Continue
WARNING: ILLEGAL DCB ADDR DCB x:xxxxxx	Specified DCB was declared an external DEF and the DEF has been defined with either a negative address or a constant.	Continue

Table 14. Overlay Loader Diagnostics (cont.)

Text	Meaning	Action
WARNING: ILLEGAL DCB NAME SEGxxxxx {ULIB / ROM / SLIB} xxx DCB x:xxxxxx	Specified DCB name is illegal and will not be included in DCBTAB. Monitor DCBs (M:) must have standard OPLB names. User DCBs (F:) must not exceed eight EBCDIC characters in length.	Continue
WARNING: LCOM name OF SIZE xxxx GREATER THAN ALLOCATED SEGxxxxx {ULIB / ROM / SLIB} xxx SEQNOxxx	The named labeled COMMON block (DSECT) with the size specified in words is greater than the size allocated.	Continue
WARNING: NO ENTRY ADDRESS FOR ROOT	Root does not have an entry address.	Continue
WARNING: OVERLAY SEG GREATER THAN 16K	Specified overlay segment exceeds the maximum size record that can be loaded by the Monitor SEGLOAD function.	Continue
WARNING: PROGRAM EXCEEDS SPECIFIED ADDR LIMITS	User's program exceeds the address limits, either specified on !OLOAD or the defaults for background/foreground programs.	Continue
WARNING: UNDEFINED DEF'S	User's program contains external DEFs that either have not been defined or have been defined with an expression the Loader cannot resolve. Map will indicate the name(s) of the undefined DEFs.	Continue
WARNING: UNDEFINED ENTRY ADDR SEGxxxxx	Expression defining the entry address for the specified segment cannot be resolved by the Loader.	Continue
WARNING: UNSATISFIED REF'S	User's program contains unsatisfied external REFs. Map will indicate the name(s) of the DEFs.	Continue
WRITING ON INPUT DEVICE DCB x:xxxxxx	A DCB that the Overlay Loader writes with has been assigned to an IN device. Either the user has assigned incorrectly or the Loader has a program error.	Abort
yyndd WRITE PROT	Specified RAD is write-protected.	Wait and 1. Reset RAD protection switches or 2. Interrupt and key in "SYC", or 3. Interrupt and key in "X" if the job is not allowed to write on protected areas of the RAD.

USER LOAD-TIME ASSIGNS

M:DCB AND F:DCB

DCBs identified by external definitions must exist in the root for each unique reference to an M:DCB or F:DCB. These are either inserted explicitly by the user or built implicitly by the Loader. A user can change DCB assignments in several ways:

1. By modifying the DCB at execution time.

2. By using a load-time :ASSIGN (foreground and background).

3. By using a run-time !ASSIGN (~~not allowed for a foreground program~~). (used by background job or foreground program run from batch stream).

RUN-TIME ASSIGNS

Run-time !ASSIGNs (by Job Control Processor) apply only to the background job in which they are inserted. Change of assignment for foreground programs is permitted only through STDLB key-ins and load-time :ASSIGNs.

LOAD-TIME ASSIGNS

Load-time :ASSIGNs are changes to the respective DCB at load-time, so that the given assignment remains as a part of the program. This effectively allows assignments for foreground programs, and assignment of DCBs with nondefault cases.

FORTRAN INTERFACE

System interface between FORTRAN-produced programs and RBM is the shared responsibility of the FORTRAN compiler-Loader-RBM complex. This complex enables the user to program real-time programs for foreground operation using Real-Time FORTRAN language without having to use symbolic coding to create the system interface (see FORTRAN job examples in Chapter 8).

Symbolic code and control information can be used to give the FORTRAN user added versatility in cases where compatibility with other FORTRAN configurations is not a factor. However, such coding is not required. That is, the user can write and execute a program to service real-time interrupts without any symbolic embellishment of the FORTRAN language and without destroying the real-time response required.

COMMON ALLOCATION

BLANK COMMON

By default, blank COMMON is allocated beginning at the end of the longest path as illustrated in Figure 11.

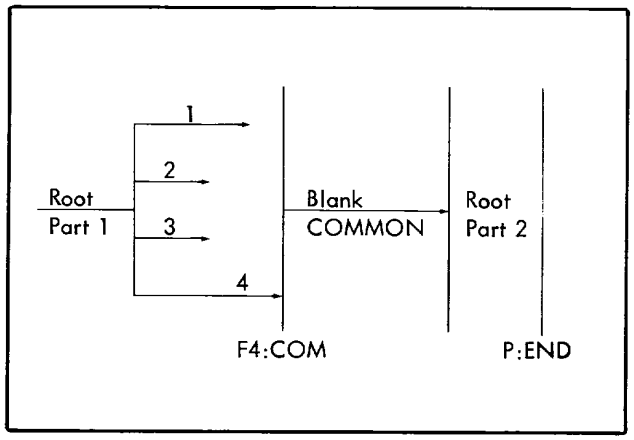


Figure 11. Blank COMMON Allocation by Default

The size of blank COMMON is determined by the size of the largest blank COMMON encountered during the loading of all segments.

An optional COMMON control command allows the user to specify that the blank COMMON base is to be set immediately following that segment, as illustrated in Figure 12.

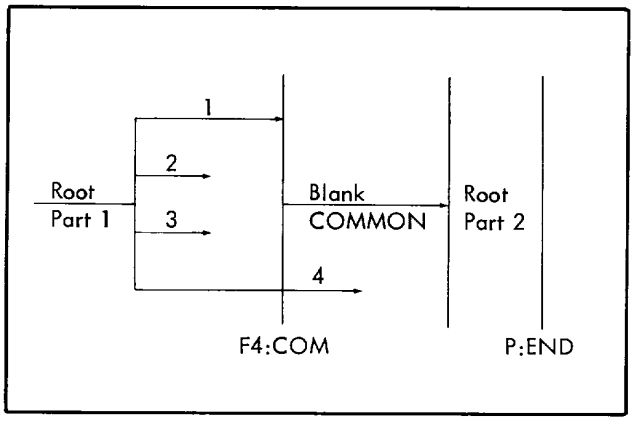


Figure 12. Blank COMMON Option

Note that in Figure 12, segment 1 sets the COMMON base so that segments 1, 2, and 3, share all COMMON, but segment 4 overlays a portion of COMMON. Thus, segments 1, 2, and 3, might operate on a large array, leaving the results in upper COMMON for segment 4, which can reclaim the remainder of the COMMON storage. However, a COMMON allocation in segment 4 would be necessary to align references to the upper portion of COMMON.

LABELED COMMON

Labeled COMMON is allocated by the Loader either by default in the segment in which the block is first encountered, or specifically, by the parameters on the LCOMMON control command. All references to a labeled COMMON block must be in the same path as the definition. Note that labeled COMMON in the root is available to all segments. A labeled COMMON block must be initialized in the segment that is allocated.

The LCOMMON control command will allocate labeled COMMON in the segment specified by the preceding SEG command. The example

```
:LCOMMON (A,100),(B,101),(XRAY,50)
:SEG (LINK,201,ONTO,0)
```

will allocate a labeled COMMON block /A/ of 100 words, a block /B/ of 101 words, and a block /XRAY/ of 50 words in segment number 201.

CONNECT

The Loader does not provide any facility for generating code to connect foreground programs to interrupts or to trigger interrupts. The CONNECT statement in FORTRAN plus the Monitor CONNECT call provides the necessary interface.

CALLING OVERLAY SEGMENTS

The Overlay Loader generates no implicit calls for loading overlay segments, and generates no explicit code for such calls. FORTRAN programs to be run in overlay form must call the FORTRAN run-time routine SEGLOAD, which calls the SEGLOAD function of the Monitor. The identification numbers in the argument list must correspond to the identification number on the SEG control command.

The SEGLOAD function calls in the overlay segment and returns, e.g.,

```
CALL SEGLOAD (I)
CALL SUBROUTINE
```

where I is the segment ident.

MAIN PROGRAM NAME AND ENTRY

The entry point of a FORTRAN main program is not necessarily the first location of the program. The compiler will output an external definition to identify it as a FORTRAN main program. The entry point for that program is either the transfer address on the main program, or the value specified with the ENTRY keyword on the :ROOT command.

LABELED COMMON NAMES

Labeled COMMON blocks are identified as DSECTs, labeled with an external definition the same as the block name.

BLANK COMMON NAMES

Blank COMMON references are identified as DSECTs with the unique external definition name F4:COM.

CORE LAYOUT AT EXECUTION TIME

The core storage area allocations for a typical segmented program are illustrated in Figure 13.

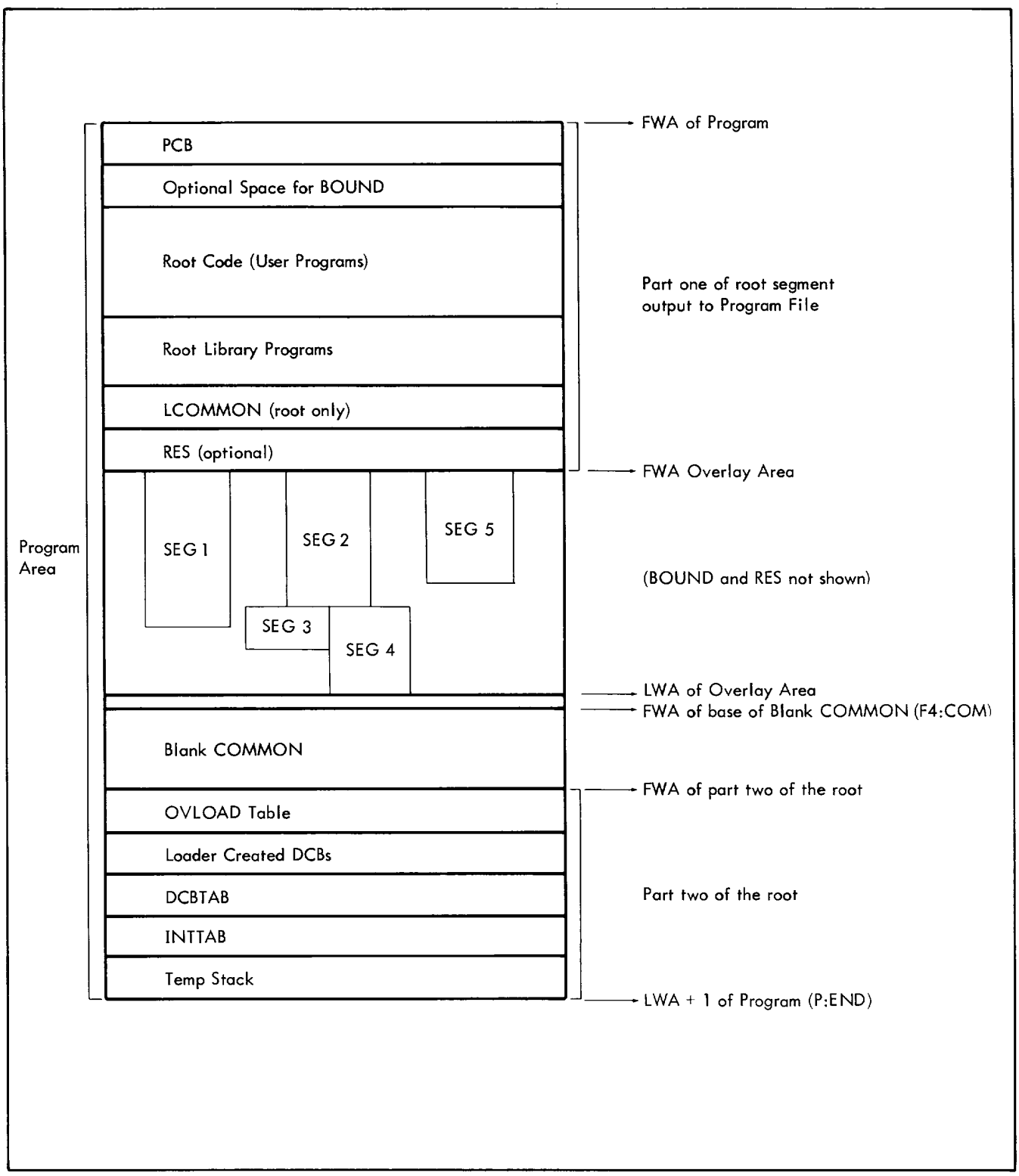


Figure 13. Standard Core Layout of a Program

7. RAD EDITOR

The RAD Editor is a background processor that performs RAD allocation for RAD areas by generating and maintaining directories for all permanent files. Through commands input by the user, the RAD Editor performs the following functions:

- Adds or deletes entries to the permanent file directories that, in turn, allocate and release permanent RAD space within a RAD area.
- Copies data files onto the RAD.
- Appends records to the end of an existing RAD file.
- Compacts permanent file directories and permanent RAD areas.
- Truncates empty space from the end of RAD files.
- Maps permanent RAD file allocation.
- Dumps the contents of RAD files or entire RAD areas.
- Copies permanent RAD files.
- Copies object modules contained in the libraries.
- Saves the contents of RAD areas on a magnetic or paper tape device in a self-reloadable form.
- Restores previously saved RAD areas to their RAD location.
- Maintains library files on RAD for use by the Overlay Loader.
- Zeros out (clears) complete RAD areas.
- Temporarily inhibits the use of bad tracks on the RAD.

OPERATING CHARACTERISTICS

FILE ALLOCATION

The RAD Editor performs RAD allocation for all permanent files. The name, size, and location of each permanent RAD area are indicated through the use of a Master Directory that is set up at system initialization in the resident portion of RBM. The permanent RAD areas maintained by the RAD Editor are

Background programs (BP).

Data (D1-DF). A maximum of 15 Background and Foreground data areas are allowed.

Foreground programs (FP) contain User Library.

System programs (SP) contain System Library.

The Editor controls file allocation by generating and maintaining a directory entry for each file within the above permanent RAD areas. Every permanent RAD area has a directory that begins in the first sector of its own area. A directory consists of entries with the following information:

- File name (maximum length of eight alphanumeric characters).
- Resident foreground program flag.
- File type; blocked; unblocked, or compressed.
- Granule size in bytes (used for direct access).
- File size (current number of records in file).
- Record size (bytes per logical record).
- Relative RAD address of the first sector defined for the file.
- Relative RAD address of the last sector defined for the file.

Before any permanent RAD file can be written, space must be allocated for the file by requesting the RAD Editor to add a new entry to the designated directory. Directory entries may be added or deleted by using RAD Editor commands. The following method is used to allocate files:

1. Permanent RAD files are allocated sequentially, beginning in the second sector of the area, with every file beginning and ending on a sector boundary.
2. A new directory entry is added as the last entry to the existing directory and the corresponding space for the file is allocated.
3. When all available space in an area is exhausted, a complete search of the file directory is made for unallocated areas made available through file deletions. The smallest area containing a sufficient amount of space to allocate for the file is selected. If sufficient space is not found upon searching the directory, the operation is aborted. To overcome this problem, RAD squeezing may be requested to recover the unused storage within a permanent RAD area by compressing the directory entries and files (see Figures 14 and 15).
4. File deletion is accomplished by zeroing out the appropriate directory entry.

SKIPPING BAD TRACKS

The method used to handle bad RAD tracks is as follows. The :BDTRACK command removes the track from use by placing a special entry in the file directory and allocating the track as a file. The :GDTRACK command returns the

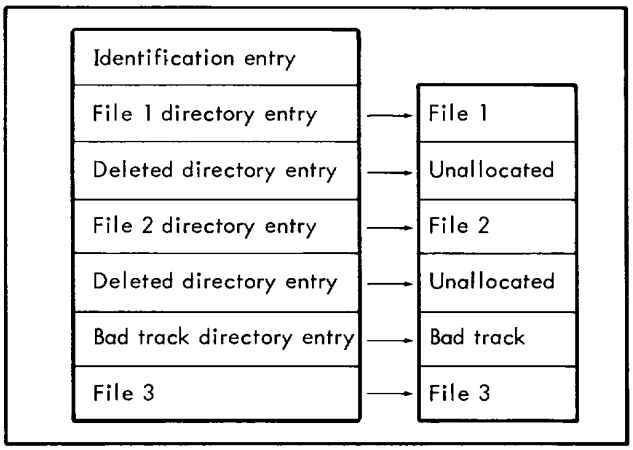


Figure 14. Permanent RAD Area Before Squeezing

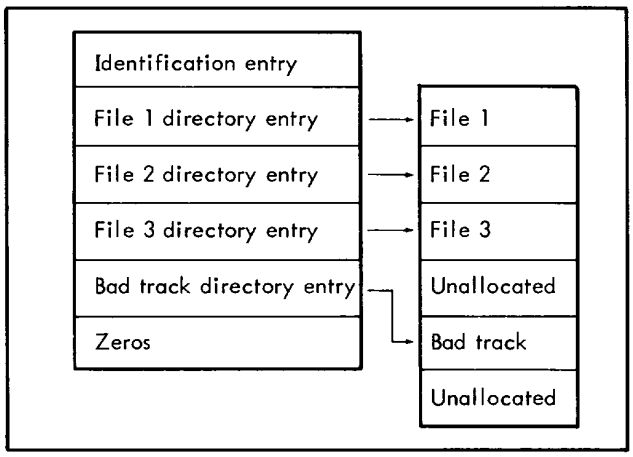


Figure 15. Permanent RAD Area After Squeezing

track for RAD Editor use by deleting the file directory entry. When a bad track is discovered, it it the user's responsibility to prevent it from being used by deleting the defective file and reallocating an area for the new file if it is to be regenerated.

SYSTEM AND USER LIBRARY FILES

System and User Library files are searched by the Overlay Loader to satisfy external references. These files are generated and maintained by the RAD Editor in a form that can be rapidly and easily searched by the Overlay Loader. The System Library files must reside in the System Programs (SP) area, and the User Library files must reside in the Foreground Programs (FP) area. Each library consists of three unblocked files: the Module Directory File (MODIR), DEFREF File (DEFREF), and EBCDIC File (EBCDIC); and one blocked file: Module File (MODULE). The user must define and allocate these library files via the RAD Editor by using the file names that appear within parentheses above and defining the files as blocked or unblocked. As an aid in approximating the file sizes, the user can use the algorithms given below.

The RAD Editor is the only processor that should write in the library files. The files are generated from information contained in the object modules read in by the RAD Editor. Each module is identified within the library files by a DEF. The first DEF encountered in the module is considered the module name, and no other DEF in a program will be so recognized. Any module may be referenced by using the first DEF in a program, and modules may be copied or deleted through its use.

ALGORITHMS FOR COMPUTING LIBRARY FILE SIZES

The following algorithms can be used to determine the approximate sizes of the four files in a library. It is not crucial that the file sizes be exact, since any unused space can be recovered via the :TRUNCATE command. The approximate number of sectors (n_{MODIR}) required in the MODIR file is

$$n_{MODIR} = \frac{3\,(i)}{s}$$

where

- i is the number of modules to be placed in the library.
- s is the RAD sector size in words.
- 3 words is the length of a MODIR file entry.

The approximate number of sectors (n_{EBCDIC}) required in the EBCDIC file is

$$n_{EBCDIC} = \frac{2\,(d)}{s}$$

where

- d is the unique number of DEFs in the library.
- s is the RAD sector size in words.
- 2 words is the average length of an EBCDIC file entry.

The number of records (n_{MODULE}) required in the MODULE file is

$$n_{MODULE} = \sum_{i=1}^{n} C_i$$

where

- n is the total number of modules in the library.
- C_i is the number of card images in the ith library routine.

The number of sectors (n_{DEFREF}) in the DEFREF file is

$$n_{DEFREF} = \frac{\sum_{i=1}^{n} 1 + \frac{d_i + r_i}{2}}{s}$$

where

n is the total number of routines in the library.

d is the number of DEFs in the ith library routine.

r is the number of REFs in the ith library routine.

s is the RAD sector size in words.

RAD AREAS PROTECTION

Updating or squeezing of permanent RAD areas containing information for real-time programs (foreground program and foreground data areas) must not occur while the foreground is utilizing these permanent RAD areas. The user must ensure that the RAD Editor is not modifying a permanent RAD area at the same time a foreground program is using it.

Software protection of the SP, FP, BP, and foreground data areas of the RAD is provided by requiring the operator to key in "SY" before any of these areas are modified by a background processor. The only areas that can be modified that do not require a SY key-in are the Background Data areas.

CALLING RAD EDITOR

When a !RADEDIT control command is read from the C device, the RAD Editor is loaded into core memory from the RAD. Control is transferred to the RAD Editor which reads commands from the C device that specify the functions to be performed.

The form of the command is

```
!RADEDIT
```

The RAD Editor is terminated when a record with an ! in column one is read from the C device (with the exception of !EOD). An !EOD indicates an end-of-data to the RAD Editor when data is input via the :COPY command.

COMMAND FORMATS

All RAD Editor commands are input from the C device and listed on LL. The general form for RAD Editor commands is identical to the RBM control command format described in Chapter 2, with the symbols below being used to aid in describing the RAD Editor commands in this chapter.

aa refers to a permanent RAD area and must be one of the following:

- BP is the Background Programs area.
- D1 through DF is the Background and Foreground Data areas.
- FP is the Foreground Programs area.
- SP is the System Programs area.

zz refers to any RAD area.

nnnnnnnn refers to a file name[†] or library module (maximum name length of eight alphanumeric characters).

yyndd refers to a physical device name, where

- yy specifies the type of device: CR, CP, etc.
- n specifies the IOP number: A for IOP0, B for IOP1, etc.
- dd specifies the device number: 03, 80, etc.

OP refers to an operational label: BI, SI, etc.

RAD EDITOR COMMANDS

:ALLOT The :ALLOT command adds a new entry to the specified permanent file directory that allocates space for a new file. After space has been allocated, files can be written by either background or foreground programs. The space allocated for the new entry is zeroed out.

The form of the command is

```
:ALLOT (FILE,aa,nnnnnnnn) [,(option)] . . .[,(option)]
```

where the options are

FORMAT,type specifies the file format where type is

- U for unblocked.
- B for blocked.
- C for a compressed file.

The default value is unblocked.

FSIZE,value specifies the decimal length of the file in logical records. The default value is 1000.

RSIZE,value specifies the decimal number of words per record. The logical record size is used in sequentially accessing a file. For a compressed file, record size is omitted and the Monitor blocks compressed files into 256-word records. Blocked files have a default value equal to 128 words per record.

[†]If this file name is RBM and in the SP area, it cannot be copied or dumped.

If the record size is greater than 128 words, unblocked organization will be given. Unblocked files have a default record size equal to the granule size.

GSIZE,value specifies granule size in words and is used for direct access only. The default size will be equal to the RAD sector size.

RF indicates that the file contains a resident foreground program and is applicable only if the FP area is specified. If RF is omitted, the file does not contain a resident foreground program. Any program flagged as resident foreground will be automatically loaded into core every time the system is booted from the RAD.

Examples:

1. An unblocked file:

```
:ALLOT (FILE,BP,TEST), (FORMAT,U), (FSIZE,50),
    (RSIZE, 90)
```

This example allocates space for the unblocked file TEST in the BP area of the RAD, with a file size of 50 records and a record size of 90 words.

2. A blocked file:

```
:ALLOT (FILE,FP,TESTA), (FORMAT,B),(FSIZE,50),
    (RSIZE,30), RF
```

This example allocates space for the blocked file TESTA in the FP area, with a record size of 30 words and a file size of 50 records. This is a resident foreground program.

:COPY The :COPY command copies single files of data or modules (EBCDIC, BINARY in standard binary format, or nonstandard binary) from one device to another. Input and output must be copied to and/or from the RAD. Files are copied until an !EOD or tape mark is encountered, except when the CC option is specified, which is terminated when an :EOD is encountered. A logical file mark will be written onto the output file.

On 7-track tape, a copy from a file to a device assigned to an op label will be in packed binary format. If a device is specified, the information (data) will be written unpacked.

When nonstandard binary (BIN) or control commands (CC) are copied from the C device, the C device must be assigned to 0 and reassigned after the copy is completed. The assignment is made when the message

!!KEYIN STDLB C,0

is typed to the operator and reassigned when the message

!!COPY ENDED

appears.

An !ATTEND card must be used to force a pause for operator intervention whenever the BIN and CC options are specified.

The general form of the command is

```
                    FROM                          TO
:COPY ( { FILE,aa,nnnnnnnn }                { FILE,aa,nnnnnnnn }
      ( { LIB,aa,nnnnnnnn  } ) , ( { LIB,aa                    } )
      ( { IN, {op}         }       { OUT, {op}                 }
      ( { IN, {yyndd}      }       { OUT, {yyndd}              }
    [,VFC] [,ADD] [,BIN] [,CC] [,FBCD]
```

where

FILE indicates either a file in a permanent RAD area, a file in the Background Temp area where nnnnnnnn is the designated file, or the Checkpoint of IOEX access area where nnnnnnnn is not applicable. Areas CK and XA are only allowed as input files.

LIB indicates a library object module(s) in the SP or FP area.

IN indicates an input operation from a non-RAD device is to be performed.

OUT indicates an output operation to a non-RAD device is to be performed.

VFC indicates vertical format control is desired on printing.

ADD indicates records are to be added to the end of an already existing file.

BIN specifies that nonstandard binary information is to be copied from the card reader or to the card punch.

CC specifies that control commands are to be copied from the C device.

FBCD specifies that BCD input is to be converted to EBCDIC.

The following are examples and explanations of the different types of copies that can be performed.

Examples:

```
:COPY (IN, {op}    ) ,(FILE,aa,nnnnnnnn)
       {yyndd}
```

This example copies a file of data onto the specified RAD file.

```
:COPY (IN, {op / yyndd}) ,(FILE,aa, nnnnnnnn),CC
```

This example copies a file of data containing control commands from the C device onto the specified RAD file.

```
:COPY (IN, {op / yyndd}) ,(FILE,aa, nnnnnnnn),ADD
```

This example adds data to the end of an already existing RAD file.

```
:COPY (IN, {op / yyndd}) ,(LIB,aa)
```

This example copies the library object modules to the specified library. The library being copied will completely replace an already existing library.

```
:COPY (IN, {op / yyndd}) (LIB,aa),ADD
```

This example adds the library object modules to the specified library.

```
:COPY (FILE,aa,nnnnnnnn),(FILE,aa,nnnnnnnn)
```

This example copies the contents of the first specified RAD file to the second specified RAD file.

```
:COPY (LIB,aa,nnnnnnnn), (OUT,{op / yyndd})
```

This example copies one library object module to the specified output device. The "nnnnnnnn" parameter is the name of the library object module to be copied.

```
:COPY (FILE,aa,nnnnnnnn), (OUT,{op / yyndd}),VFC
```

This example lists the contents of an EBCDIC file with vertical format control.

```
:COPY (FILE,aa,nnnnnnnn), (OUT,{op / yyndd})
```

This example copies the contents of the specified RAD file onto the specified device.

```
:COPY (FILE,aa,nnnnnn), (OUT,{op / yyndd}),BIN
```

This example copies nonstandard binary from the specified RAD file to the card punch.

```
:COPY (FILE,aa), (OUT,{op / yyndd})
```

This example copies the contents of the IOEX access (XA) or Checkpoint (CK) areas to the specified output device.

:DELETE The :DELETE command deletes either a file directory entry and file from a specified permanent RAD area, or an object module from the designated library. The space formerly allocated is not used until a :SQUEEZE is executed.

The :DELETE command has the form

```
:DELETE ({LIB / FILE} ,aa,nnnnnnnn )
```

Examples:

1. Delete a file:

```
:DELETE (FILE,BP,TESTA)
```

2. Delete an object module:

```
:DELETE (LIB,SP,CSCN)
```

This example specifies that an object module named CSCN is to be deleted from the Library in the SP area.

:CLEAR The :CLEAR command zeros out the specified RAD areas which results in deleting all files and file directories in the area.

The form of the command is

```
:CLEAR zz,zz,. . .
```

where zz is any RAD area.

Example:

```
:CLEAR D1,DF
```

This example specifies that permenent RAD areas D1 and DF are to be zeroed out.

:SQUEEZE The :SQUEEZE command regains unused space within permanent RAD areas resulting from file deletions and truncations and library module deletions. Unused space is regained by compressing file directory entries and their associated files, and library file entries and their associated library modules. Within the libraries, the Module Directory File (MODIR) and the Module File (MODULE) entries and modules are compressed to regain the unused space. Space is regained in the remaining two files, EBCDIC File (EBCDIC) and DEFREF File (DEFREF), by regenerating them completely from the Module Directory and Module Files.

The forms of the command are

1.
```
:SQUEEZE aa,aa,aa,. . .
```

2.
```
:SQUEEZE ALL
```

Examples:

1. Regain unused space in specified area:

```
:SQUEEZE SP
```

This example regains unused space between files and between modules in the SP area only.

2. Regain space in all permanent RAD areas.

```
:SQUEEZE ALL
```

This example regains unused space between all files and modules in all permanent RAD areas.

:TRUNCATE The :TRUNCATE command is used to truncate empty space from the end of specified file(s). If the allocated RAD space for a file is greater than the actual length of the file, a considerable amount of space may be left empty. This command will set the allocated space equal to the actual length of the file. For a direct access file, the length of the file in granules must be specified (g) as actual file length is unknown.

The forms of the command are

1.
```
:TRUNCATE (FILE,aa,nnnnnnnn),
          (FILE,aa,nnnnnnnn). . .,(FILE,aa,nnnnnnnn)
```

2.
```
:TRUNCATE (FILE,aa,nnnnnnnn,g),
          (FILE,aa,nnnnnnnn,g). . .,(FILE,aa,nnnnnnnn,g)
```

3.
```
:TRUNCATE aa,aa,aa,. . .
```

Examples:

1. Truncate allocated file:

```
:TRUNCATE (FILE,BP,TEST)
```

This example truncates empty space from the end of the allocated file TEST in the BP area by setting the allocated size equal to the actual size of the file.

2. Truncate all files:

```
:TRUNCATE BP,D2,D3
```

This example truncates all files in the BP, D2, and D3 areas.

:MAP The :MAP command maps the specified permanent RAD areas to the LO device (using the M:LO DCB). The map contains

1. Information from the Master Directory, consisting of the RAD, write protection, area identification, and its beginning and ending RAD addresses.
2. Information from the Permanent File Directories concerning each file in the area; file name, format,

beginning file address, ending file address, file size, record size, granule size, and resident foreground program indicator.

3. Information about object modules in the library files, consisting of the name of each module, its relocatable length, and the definitions and references in the module.

The forms of the command are

1.
```
:MAP aa,aa,aa, . . .
```

2.
```
:MAP ALL
```

Examples:

1. Map specified permanent RAD areas:

```
:MAP BP,D4
```

This example outputs a map of the permanent RAD areas BP and D4 to the LO device.

2. Map all permanent RAD areas:

```
:MAP ALL
```

This example outputs a map of all permanent RAD areas to the LO device.

An example of an ALL version of a RAD map (foreshortened for brevity) is illustrated in Figure 16.

:DUMP The :DUMP command dumps, in hexadecimal, the designated random or sequential access file onto the LO device (using the M:LO DCB). All permanent RAD areas plus the IOEX Access area (XA), Background Temp area (BT), and Checkpoint area (CK) can be dumped. The RAD Editor will sequentially access the designated file or area to be dumped.

The forms of the command are

1.
```
:DUMP (FILE,aa,nnnnnnnn) [,(SREC,value)]
      [,(EREC,value)]
```

where

aa also includes the XA, BT and CK RAD areas, and nnnnnnnn is any of the files.

SREC,value specifies the starting record (in decimal) to begin the dump.

EREC,value specifies the last record to be dumped.

2.
```
:DUMP zz[,(SREC,value)] [,(EREC,value)]
```

where

zz is any RAD area.

SREC,value specifies the starting sector (in decimal) to begin the dump.

EREC,value specifies the last sector to be dumped.

Examples:

1. Dump specified file:

```
:DUMP (FILE,BP,TEST)
```

This example specifies that the TEST file in the BP area is to be dumped onto the LO device.

2. Dump specified records:

```
:DUMP (FILE,BP,TEST),(SREC,10),(EREC,20)
```

This example specifies that records 10 through 20 of the TEST file in the BP area are to be dumped onto the LO device.

3. Dump specified sectors:

```
:DUMP BP,(SREC,6),(EREC,9)
```

This example specifies that sectors 6 through 9 of the BP area are to be dumped onto the LO device.

4. Dump all of specified RAD area:

```
:DUMP BP
```

This example specifies that all of the BP area is to be dumped onto the LO device.

```
JOB          VERSION A01

 ATT
 RADEDIT
:ALLOT (FILE,D1,TEST),(FORMAT,C),(FSIZE,10)
:MAP ALL
RAD AREA SP  RAD DCAFO  BOA 0001  EOA 1599  WP N

NAME       FORMAT  GSIZE  RSIZE  FSIZE  BOF   EOF
RBM           U    0360   0360   021₈   0001  02,9
RADBOOT       U    0360   0360   0001   0220  0220
RADEDIT       U    0360   0360   0100   0221  0320
OLOAD         U    0360   0360   0000   0321  0480
FORTRANH      U    0360   0360   0000   0481  0645
MACRSYM       U    0360   0360   0000   0646  0745
RAD AREA FP  RAD DCAFO  BOA 1600  EOA 2399  WP N

NAME       FORMAT  GSIZE  RSIZE  FSIZE  BOF   EOF
MODIR         U    0360   3132   000-   0001  00,5
EBCDIC        U    0360   2629   0001   0016  0030
DEFREF        U    0360   3644   0001   0031  0050
MODULE        B    0360   0120   0703   0051  0425

MAP OF LIBRARY IN FP AREA

 MODULE NAME   LOCATION   DEFS                    REFS
 TEST          0000       TEST                    M:SI      F:5

 ALOG          0011       ALOG                    8TO       9ALOG     9SETUP1

 DLOG          0012       DLOG                    8TO       9DLOG     9SETUP1

 EXP           0013       EXP                     8TO       9EXP      9SETUP1

 DEXP          0014       DEXP                    8TO       9DEXP     9SETUP1

 SIN           0015       SIN                     9SETUP1   8TO       9SIN

 DSIN          0016       DSIN                    8TO       9DSIN     9SETUP1

 ATAN          0017       ATAN                    8TO       8T1       9ATAN1
                                                  9ATAN2    9SETUPM

 DATAN         0019       DATAN                   8TO       8T1       9DATAN1
                                                  9DATAN2   9SETUPM

 SQRT          0021       SQRT                    9SETUP1   8TO       9SQRT

 DSQRT         0022       DSQRT                   8TO       9DSQRT    9SETUP1

 SINH          0023       SINH                    9SETUP1   8TO       9SINH
RAD AREA BP  RAD DCAFO  BOA 2400  EOA 3199  WP N

AREA BP CONTAINS NO FILES
RAD AREA BT  RAD DCAFO  BOA 4436  EOA 8191  WP B

RAD AREA XA  RAD DCAF1  BOA 4096  EOA 8191  WP X

RAD AREA CK  RAD DCAFO  BOA 4208  EOA 4435  WP M

RAD AREA D1  RAD DCAFO  BOA 3200  EOA 3679  WP B

NAME       FORMAT  GSIZE  RSIZE  FSIZE  BOF   EOF
TEST          C    0360   1024   0000   0001  0003
FIN
TOTAL JOB TIME=00:01:00
 BEGIN IDLE
```

Figure 16. ALL Map Example

:SAVE The :SAVE command saves the specified RAD area(s) on the BO device (using the M:BO DCB) for subsequent restoration. The BO device must be either a magnetic-tape or paper-tape device. The image of the designated RAD area(s) and the RBM bootstrap are written on BO in self-reloadable format. The BO output contains a bootstrap loader, followed by the RAD image of the RBM bootstrap, and the designated area(s). Sectors containing all zeros are suppressed. Executing the bootstrap loader causes the RAD image to be read into memory and restored onto the RAD(s) without RBM control. The BO output can also be used to restore the RAD via the :RESTORE command. If the BO device is a magnetic tape, the tape is rewound and the data saved is verified. If the BO device is a paper tape, the paper tape must be input on the BI device for verification. If the tape verifies correctly, the message

'SAVE TAPE OK'

is output.

The forms of the command are

1.
```
:SAVE zz,zz,...
```

where zz can be any RAD area.

2.
```
:SAVE ALL
```

where ALL includes all RAD areas except Background, Temporary, and Checkpoint.

Examples:

1. Dump specified areas to secondary storage:

```
:SAVE SP,BP,D2
```

This example specifies that RAD areas SP, BP, and D2, with a preceding bootstrap loader, are to be saved on the BO device for subsequent reloading.

2. Dump all RAD areas to secondary storage:

```
:SAVE ALL
```

This example specifies that all RAD areas, with a preceding bootstrap, are to be saved on the BO device for subsequent reloading.

:RESTORE The :RESTORE command restores the specified permanent RAD areas that were saved by the :SAVE command. Input is read from the BI device (using the M:BI DCB), and the bootstrap is ignored. Read after write is employed to verify the data restored.

The form of the command is

```
:RESTORE zz,zz,...
```

Example:

```
:RESTORE SP,BP,D2
```

This example specifies that the RAD areas SP, BP, and D2 (previously saved with a :SAVE directive) are to be restored.

:BDTRACK The :BDTRACK command specifies the RAD and the hexadecimal track numbers that are not to be used by the RAD Editor. A track containing a sector of the file directory is not permitted to be removed from use.

The form of the command is

```
:BDTRACK  yyndd,number[,number]...
```

Example:

```
:BDTRACK DCAFO,10,11
```

This example specifies that the RAD Editor is to be inhibited from using tracks 10 and 11 on the RAD DCAF0.

:GDTRACK The :GDTRACK command specifies the RAD and the hexadecimal track numbers that now can be used by the RAD Editor. The tracks were previously removed from use by the :BDTRACK command.

The form of the command is

```
:GDTRACK yyndd,number[, number]...
```

Example:

```
GDTRACK  DCAF0,10,11
```

This example specifies that previously inhibited tracks 10 and 11 are to be restored for use by the RAD Editor.

ERROR MESSAGES

The RAD Editor outputs error messages on the OC and LL devices. If OC and LL are assigned to the same device, duplication of messages on LL is suppressed. If an operator response is required, the RAD Editor will call the Monitor "WAIT" routine. The operator initiates a console interrupt and keys in one of the following commands to the Monitor.

C Continue and read next record from the C device.

X Abort RAD Editor and return control to Monitor.

COC Continue and read a record from the OC device (used only in conjunction with the error message "ERROR ITEM xx").

If the Editor aborts because of an irrecoverable I/O error, the physical device name is included in the abort message.

The error messages output by the RAD Editor and their meanings are given in Table 15.

RAD RESTORATION MESSAGES

The messages itemized in Table 16 are written on the keyboard/printer during RAD restoration via the bootstrap loader produced by SAVE. Unless otherwise specified, the computer will go into a WAIT after writing a message.

Table 15. RAD Editor Error Messages

Message	Meaning	Action Taken
AREA xx CANNOT CONTAIN A RESIDENT FOREGROUND PROGRAM	Illegal area specified. Only the FP area can contain a resident foreground program	Operation is aborted.
AREA xx CKSM ERROR	A checksum error exists on the RAD SAVE tape in the specified area.	Operation is aborted.
AREA xx CONTAINS NO FILES	Specified area contains no files.	Editor continues.
AREA xx INCOMPATIBILITY	Attempting to restore specified area onto a different type of RAD from which it was saved, or the area to be restored is too large for the same area using the current Master Directory.	Operation is aborted.
AREA xx IS NOT ALLOCATED	Specified area was not allocated at SYSGEN.	Operation is aborted.
AREA SPECIFIED DOES NOT CONTAIN A LIBRARY	An area other than SP or FP was specified that does not contain a library.	Operation is aborted.
AREA SPECIFIED IS NOT MAINTAINED BY THE RAD EDITOR	An attempt has been made to use area CK, XA, or BT which is not maintained by the RAD Editor.	Operation is aborted.
AREA xx TRUNCATED	Specified area being restored is larger than the same area using the current Master Directory, but the data that was lost contained all zeros.	Operation continues.
BUFFER SMALLER THAN DATA READ	Data read exceeds the amount of available buffer space.	Operation is aborted.
CKSM ERR ON SAVE TAPE	A checksum error has been encountered while verifying the RAD SAVE tape.	Operation is aborted.

Table 15. RAD Editor Error Messages (cont.)

Message	Meaning	Action Taken
CKSM ERROR	Last record in the object module being read has a checksum error.	If the operator response is C, the Editor reads the next record from the specified device.
DUPLICATE DEF xxxxxxxx	Relocatable Object Module being copied to the library contains duplicate definitions.	RAD Editor skips to the end of the module. A key-in of C causes the Editor to read the next record from the specified device.
DUPLICATE FILE	An attempt has been made to allocate a file using a name which already exists.	Operation is aborted.
EOT on {yyndd / area, name	Unexpected end-of-tape was encountered on the specified device or file.	Operation was aborted
ERROR ITEM xx	Item number xx on the command is in error.	If the operator response is C, the Editor reads the next record from the C device. If the operator response is COC, the next record is read from the OC device. This will enable operator to rectify a directive error.
FILE xxxxxxxx WAS NOT TRUNCATED. FSIZE = 0	File was not truncated because file size being 0 suggests either a direct access file or a file with 0 records.	Editor continues
ILLEGAL BINARY RECORD	An illegal binary record (first byte not X'1C', X'3C') has been read with an object module.	If the operator response is C, the Editor reads the next record from the specified device.
ILLEGAL FILE NAME	An attempt has been made to allocate a file using GO, OV, or X1-X9 as a file name.	Operation is aborted.
ILLEGAL LOAD ITEM xx	Relocatable Object Module to the library contains an illegal load item.	RAD Editor skips to the end of the module. A key-in of C causes the Editor to read the next record from the specified device.
ILLEGAL OPTION xxx	Option specified is not permitted on a :COPY command.	Operation is aborted.
ILLEGAL USE OF :COPY	The specified combination of input and output devices on the :COPY command is prohibited.	Operation is aborted.
INVALID RSIZE. UNBLOCKED ORGANIZATION GIVEN	Maximum record size for a blocked file has been exceeded. Unblocked orgnization given.	Editor continues.
KEY ERR	Operator key-in is erroneous.	Key-in has to be either C, COC, or X.
NOT ENUF BACKG SPACE	Insufficient background space to perform the requested operation.	Operation is aborted.

Table 15. RAD Editor Error Messages (cont.)

Message	Meaning	Action Taken
RAD OVERFLOW	Allocating the amount of RAD storage indicated by the "file" parameter on the :ALLOT command would cause the permanent RAD area indicated by the "directory" parameter to overflow.	Operation is aborted.
RECORD SIZES DIFFER ON INPUT AND OUTPUT FILES	Record sizes differ on copying from RAD file to RAD file.	Operation is aborted.
REFERENCES TO F4:COM NOT ALLOWED	An external definition or reference F4:COM encountered in a Relocatable Object Module being copied to the library.	RAD Editor skips to the end of the module. A key-in of C causes the Editor to read the next record from the specified device.
ROM DOES NOT CONTAIN A DEF	Relocatable Object Module being copied does not contain an external definition.	A key-in of C causes the Editor to read the next record from the specified device.
SAVE TAPE OK	RAD SAVE tape has been verified correctly.	No action.
SEQ ERROR	Last record in the object module being read has a sequence error.	If the operator response is C, the Editor reads the next record from the specified device.
SPECIFIED FILE DOES NOT EXIST	File does not exist within the specified area.	Operation is aborted.
SPECIFIED ROM DOES NOT EXIST	Relocatable Object Module does not exist within the specified library.	Operation is aborted.
SREC VALUE GREATER THAN EREC VALUE	Parameter error on the :DUMP command. The last record to be dumped precedes the initial record to be dumped.	Operation is aborted.
TRACT xxxxx CANNOT BE DELETED	Illegal attempt to remove a track from use containing a sector of the file directory. Removal would prevent accessing of files or other sectors of the directory.	Operation is aborted.
UNABLE TO FIND AREA xx	Specified area cannot be found on the RAD SAVE tape during a :RESTORE operation.	Operation is aborted.
yyndd WRT PROT	Specified RAD is write-protected.	Operator should take appropriate action: interrupt and key in "SYC" or reset the appropriate RAD protection switches. Or, if the job is not allowed to write on protected areas of the RAD, interrupt and key in "X" to abort.

Table 16. RAD Restoration Messages

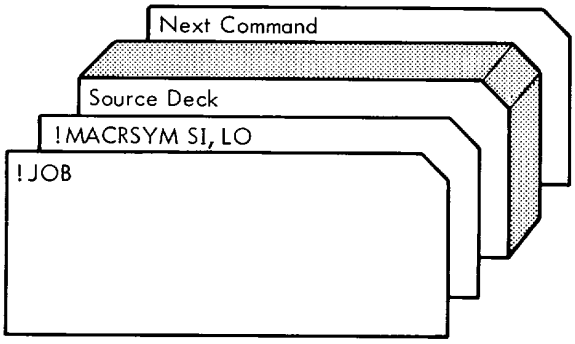
Message	Meaning	Resulting Action
CKSM ERROR	A checksum error has occurred in reading the SAVE tape.	If the WAIT condition is cleared, the bootstrap loader continues and accepts the bad record.
RAD RESTORED OK	The RAD restoration has been successfully completed.	Control is transferred from the RAD bootstrap.
TRK = xxxx DATA = ALL ZEROS	Specifies the contents of the RAD controller address register in hexadecimal at the time of a check write error.	If the data being written contains all zeros, this information is output. If the WAIT condition is cleared, the bootstrap loader continues.
yyndd ERROR, SB = xxxx	A parity or transmission error has occurred on device yyndd. Both the device status byte and operational status byte are displayed following "SB=".	There is no recovery.
yyndd UNRECOG., SB = xxxx	An unrecognized status has been returned from the indicated device. Both the device status byte and operational status byte are displayed following "SB=".	Upon clearing the WAIT condition, the operation is retried.
yyndd UNUS. END, TDV = xxxx	An unusual end status has been returned from the specified device. Both the TDV status byte and operational status byte are displayed following "SB=".	There is no recovery on a read operation. On a write operation, the write is tried again after the WAIT is cleared.
yyndd WRT PROT	The RAD is write-protected.	Program will attempt the RAD write after an SY key-in.

8. PREPARING THE PROGRAM DECK

The following examples show some of the ways program decks may be prepared for RBM operation. Unless stated otherwise, standard default cases for device assignments are assumed.

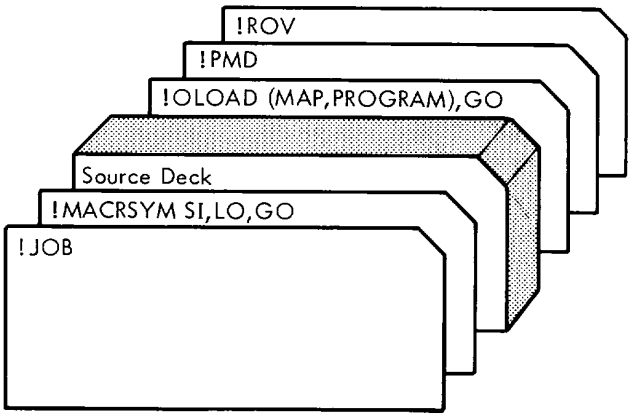
MACRO-SYMBOL EXAMPLES

ASSEMBLE SOURCE PROGRAM, LISTING OUTPUT



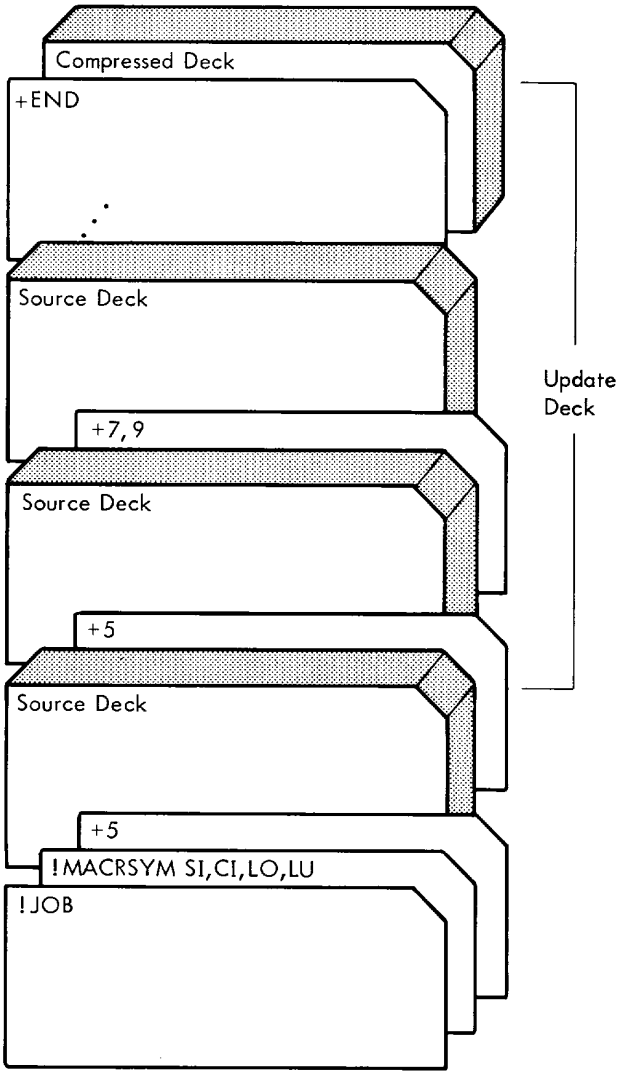
In this example, the symbolic input is received from the SI device and the listing output is produced on the LO device.

ASSEMBLE SOURCE PROGRAM, LISTING OUTPUT, LOAD AND GO OPERATIONS



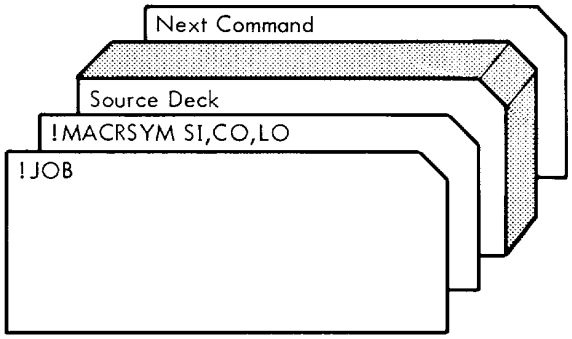
In this example, the binary object program produced from the assembly is placed in a temporary (GO) file from which it is later loaded and executed. The resultant file is always temporary and cannot be retained from one job to another. The Overlay Loader loads the program root into the OV file for execution. A postmortem dump is specified.

ASSEMBLE FROM COMPRESSED DECK WITH SOURCE AND UPDATES, LISTING OUTPUT



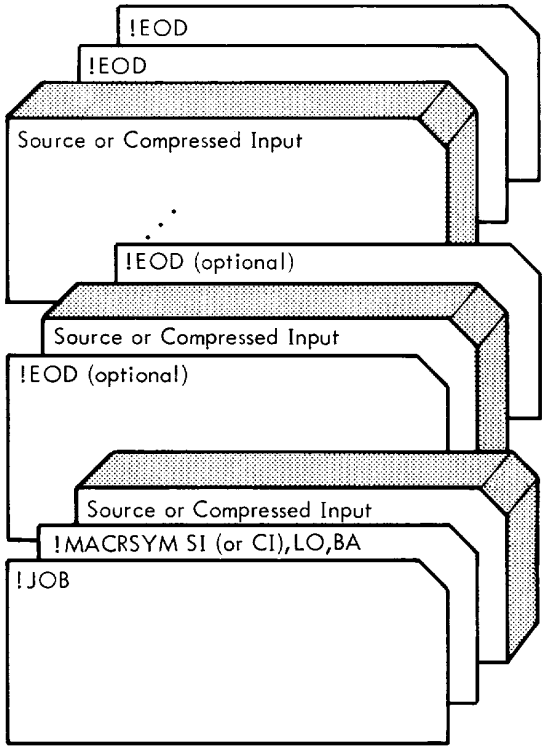
In this example, the compressed input (deck) is received from the CI device, listing output is produced on the LO device, and listing of the update deck is also produced on the LO device. The update deck is enclosed in the bracket.

ASSEMBLE SOURCE PROGRAM, COMPRESSED OUTPUT ON CARDS, LISTING OUTPUT



In this example, the compressed card output is procuded on the CO device.

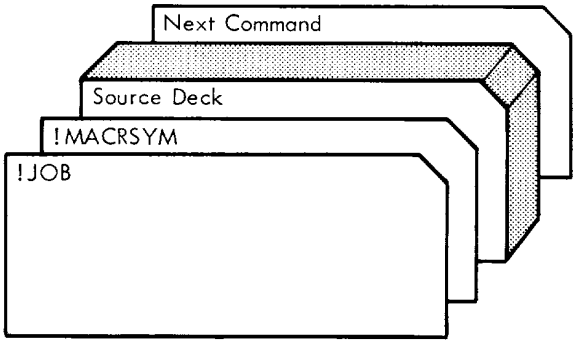
ASSEMBLE SOURCE OR COMPRESSED PROGRAM IN BATCH MODE, LISTING OUTPUT



In this example, successive assemblies are performed with a single MACRSYM command until a double EOD is read. The device assignments and options on the MACRSYM command apply to all assemblies within the batch. A program is considered terminated when an END Macro-Symbol directive is processed.

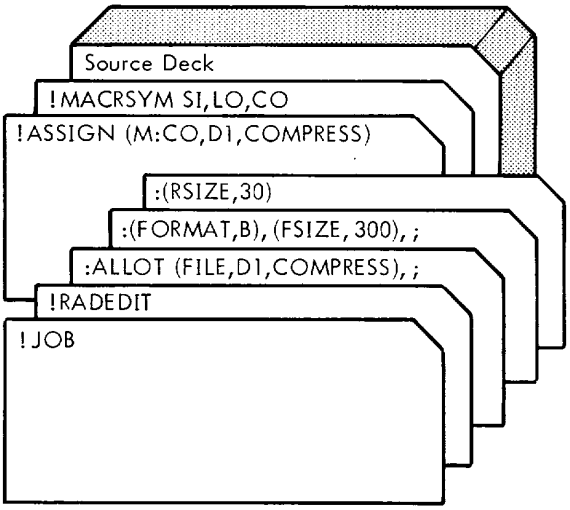
When batch assemblies consist of successive updates from card input to compressed programs from the RAD or tape, the updates are terminated by a +END card and should not be separated by !EOD cards. There must be a one-to-one correspondence of update packets to compressed programs. End-of-job is signaled by end-of-file conventions applied to the CI device.

ASSEMBLE SOURCE PROGRAM, BINARY OUTPUT ON CARDS, LISTING OUTPUT



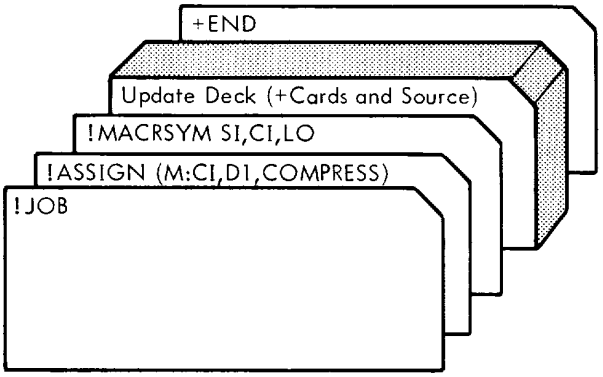
In this example, the SI, LO, and BO assignments are assumed by default.

ASSEMBLE SOURCE PROGRAM, COMPRESSED OUTPUT ON RAD FILE, LISTING OUTPUT



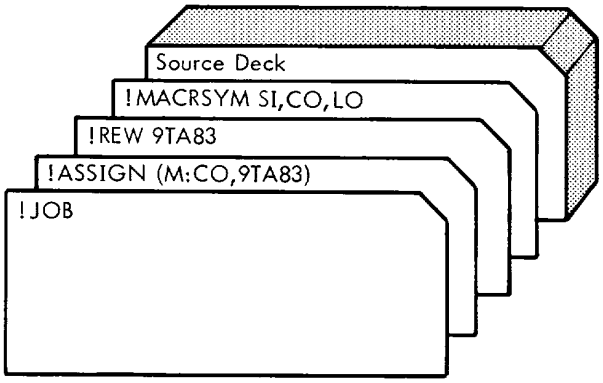
In this example, the CO device is assigned to a RAD file called COMPRESS in a background data area of the RAD. The compressed output is written on the COMPRESS file.

ASSEMBLE COMPRESSED DECK FROM RAD FILE, SOURCE UPDATES FROM CARDS, LISTING OUTPUT



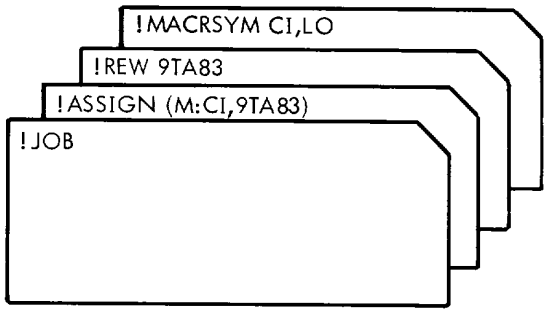
In this example, the CI (compressed input) device is assigned to the COMPRESS file in a background area of the RAD. The source update deck will be read from the SI device. In effect, this will update the assembly given in the previous example.

ASSEMBLE SOURCE PROGRAM, WRITE COMPRESSED OUTPUT ON 9-TRACK TAPE, LISTING OUTPUT



In this example, the CO device is assigned to the designated 9-track magnetic tape unit to receive the compressed output.

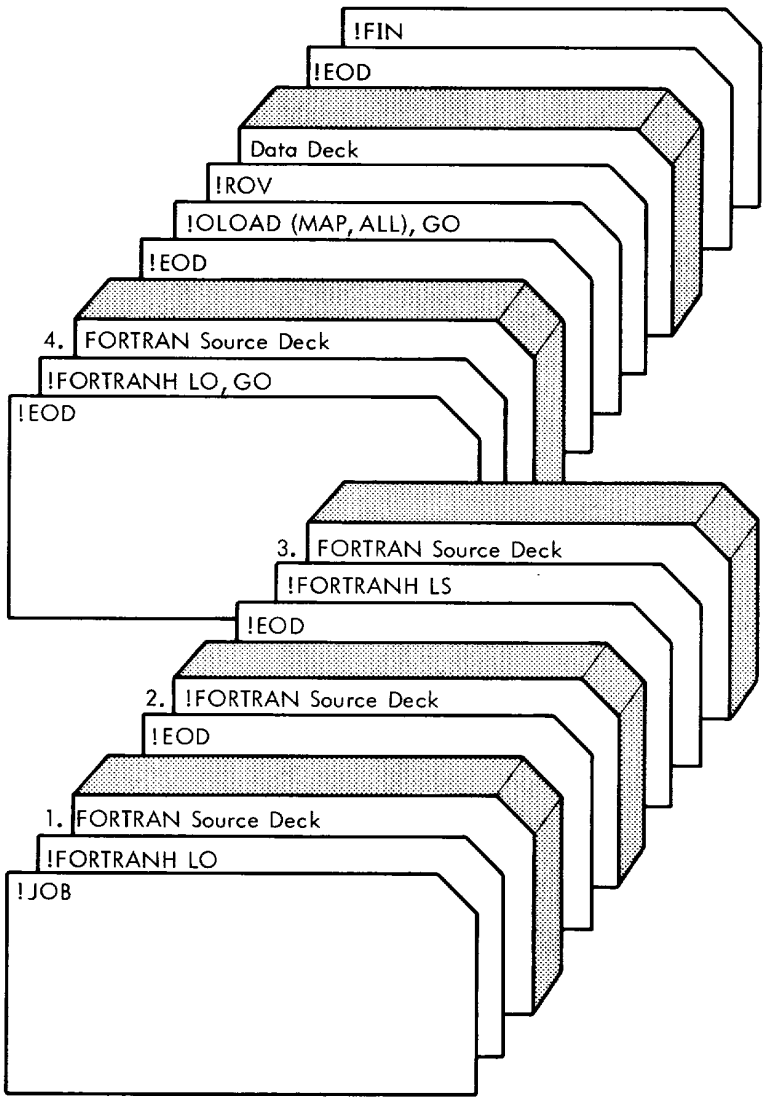
ASSEMBLE COMPRESSED PROGRAM FROM 9-TRACK TAPE, LISTING OUTPUT



In this example, the CI device is assigned to the designated magnetic tape to read the compressed input to be assembled. This is the next logical job step to follow the previous example.

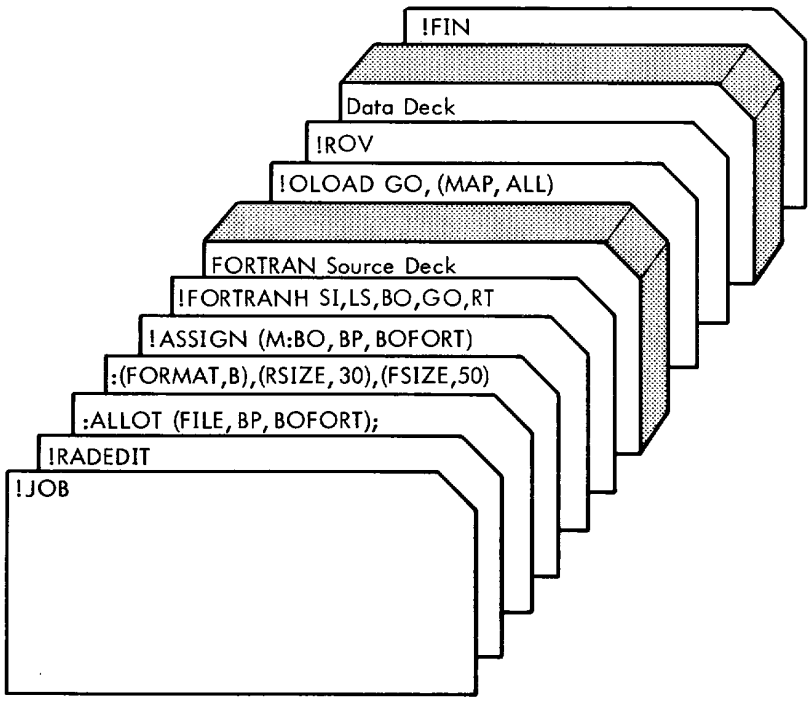
FORTRAN JOB EXAMPLES

COMBINED FORTRAN COMPILATIONS, PLUS FORTRAN COMPILE AND EXECUTE



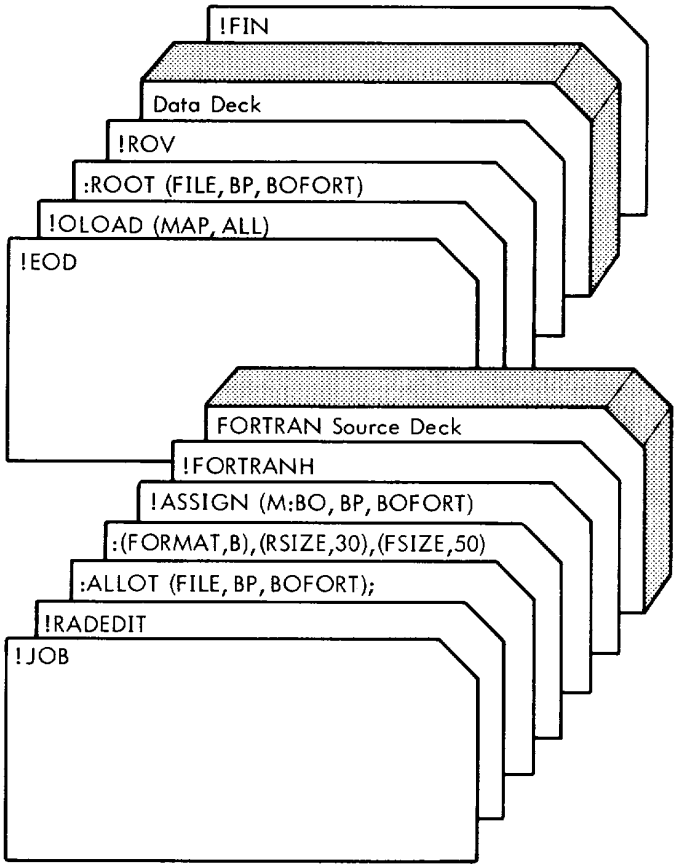
In this example, the first two source decks are compiled with mixed (source and object language) listed output. The next source program (3) is compiled with a source listing (only) being output. The final source deck (4) will compile with the object module being output on both the BO and GO files. The Loader inputs the object module from the GO file to form the Root and outputs the executable program to the OV file. The program (called OV) is executed via the !ROV command.

COMPILE AND EXECUTE FORTRAN SOURCE PROGRAM WITH REAL-TIME LINKAGES



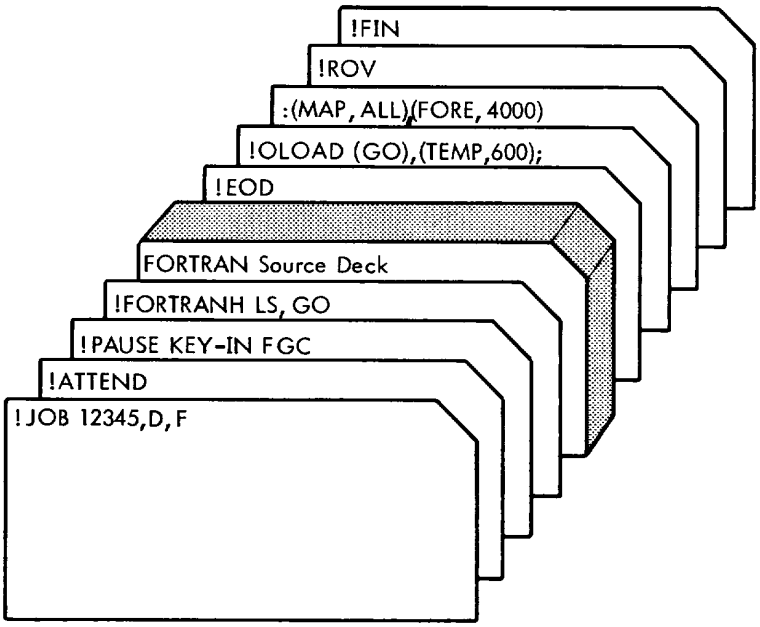
In this example, the RAD Editor allots a file called BOFORT in binary format to the Background Programs area of the RAD. The specified record size is 30 words; the file size is 50 records. The !ASSIGN command assigns the M:BO DCB (binary output) to file BOFORT. The !FORTRANH command specifies that symbolic input is to be read from the SI device, the source input is to be listed (under RBM, LS would be redundant if LO was specified in addition to LS), and the binary output is to be written out on both the BO and GO files. The relocatable binary output on the file "BOFORT" may be used as input to the Overlay Loader at a later date. Since the real-time option (RT) is specified, the linkage will be set up for real-time subroutines. The Overlay Loader (!OLOAD) reads input from the GO file, outputs an ALL map, searches the System Library (by default), and writes the executable program into the OV file. The program is run via the !ROV (Run OV) command. Note that a program with real-time linkages can also run in the background in nonreal-time mode.

COMPILE AND EXECUTE PROGRAM USING LS, BO DEFAULT OPTIONS



In this example, the LS (list source) and BO (binary output) are assumed by default on the FORTRANH command. The System Library is searched via the default option on the !OLOAD command.

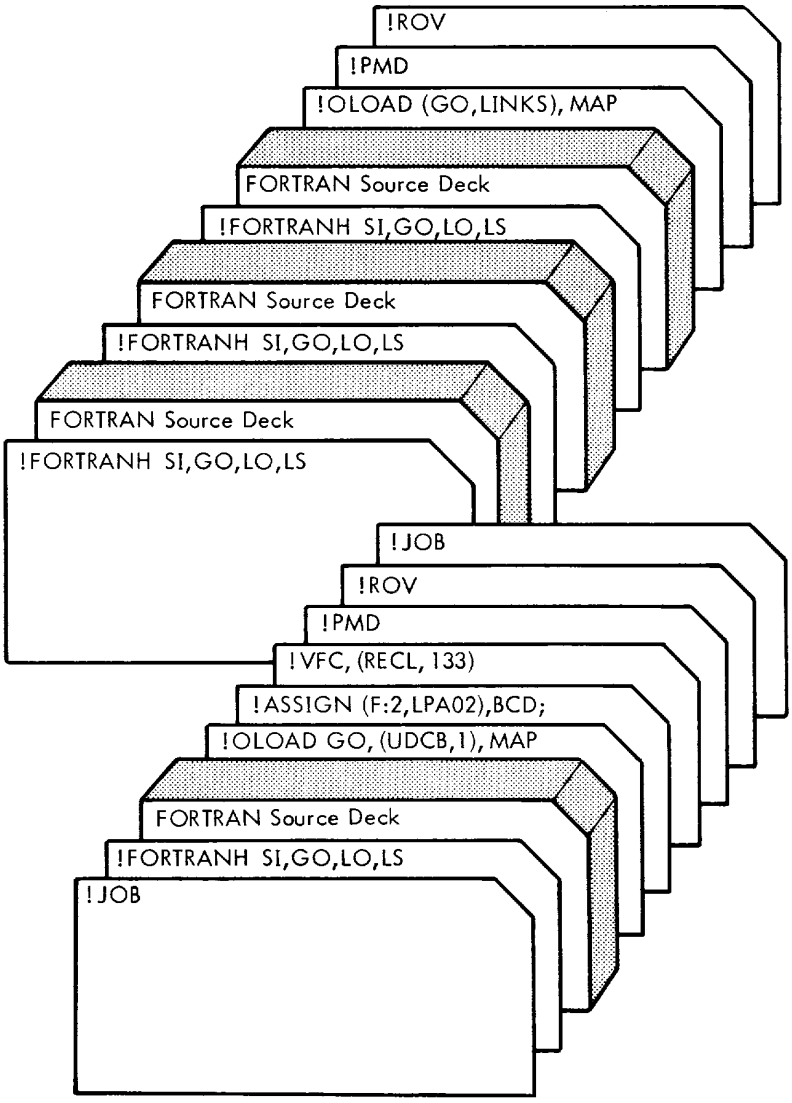
COMPILE A FORTRAN PROGRAM AND SETUP FOR EXECUTION IN FOREGROUND AREA



In this example, the !ATTEND command will inhibit the Monitor ABORT routine so that corrective action can be taken at the console in case of error. The FORE option on the !OLOAD command gives the FWA of the program in the Foreground area of memory.

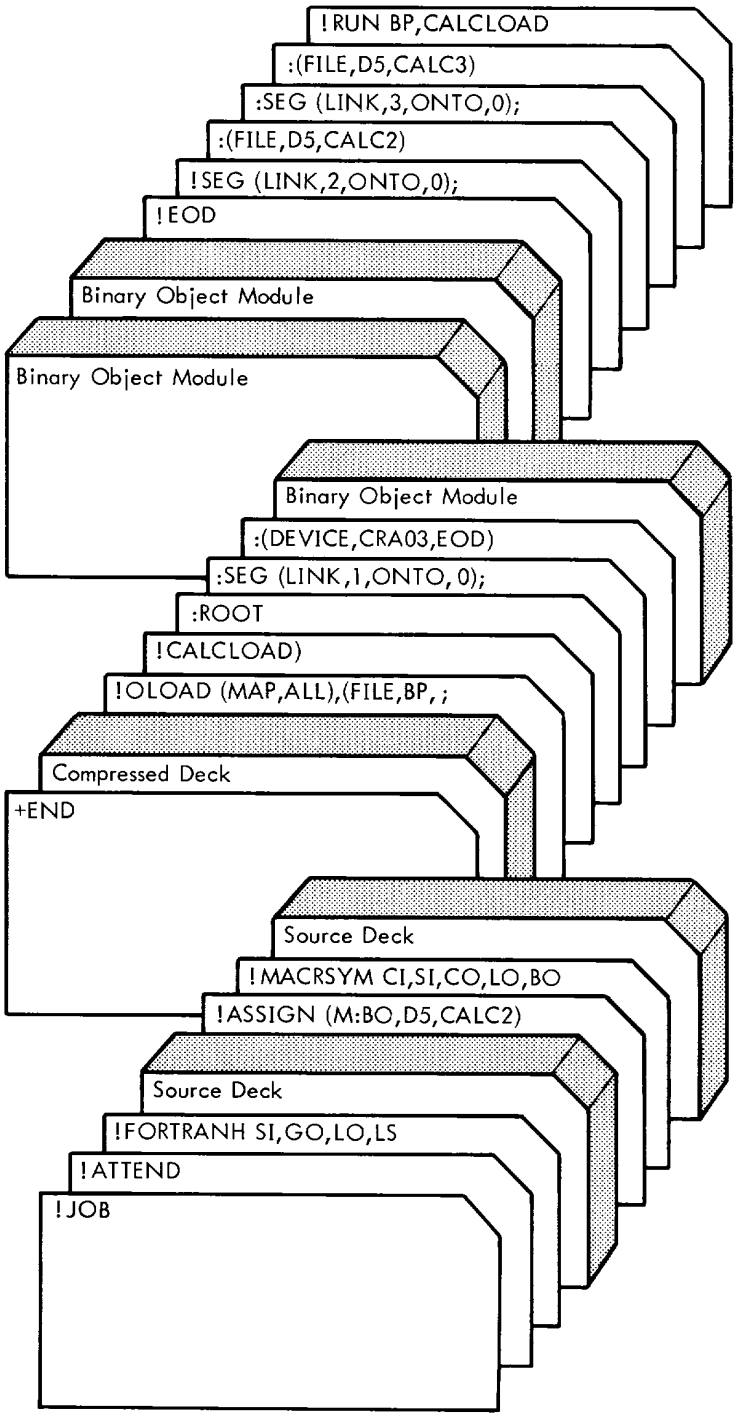
OVERLAY LOADER EXAMPLES

BATCH, USING GO LINKS



In this example, the GO file is rewound by the initial !JOB command for the first FORTRAN compilation. The Overlay Loader loads from the GO file to form a root and outputs on the OV file for execution. A SHORT map will be output. A postmortem dump is requested if the background aborts. The next !JOB command rewinds the GO file and three FORTRAN jobs are compiled, with the binary object modules output on GO to form ROM1,ROM2,ROM3. The Overlay Loader loads the first ROM for the root, the second ROM for segment 1, and the third ROM for segment 2. Note that :SEG cards are not required. The programs are executed from the OV file. A SHORT map is output. A postmortem dump is specified in case an abort occurs.

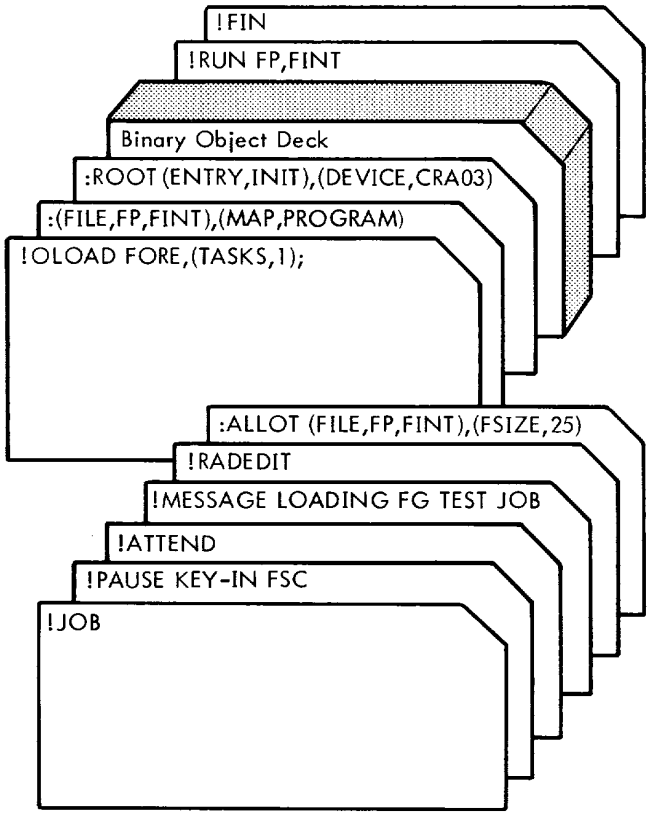
SEGMENTED BACKGROUND JOB



In this example, the JOB card rewinds the GO file, the FORTRAN source deck is compiled, and the binary object module is output on GO. The Macro-Symbol compressed source deck is updated and the binary object module is output to file CALC2 in the D5 area (previously allocated by the RAD Editor). The ROMs designated on the :ROOT and :SEG commands are loaded, and the loaded program is output to CALCLOAD in the BP area. The :ROOT command causes the ROM created by FORTRANH to be loaded from the GO file and creates the Root. The ROMs following the first :SEG command are loaded until !EOD is encountered and segment 1 is then created. The next :SEG command loads the ROM assembled by Macro-Symbol on the CALC2 file in the D5 area and creates segment 2. The last :SEG command loads one ROM from the CALC3 file in the D5 area (ROM previously created by an assembly or compilation). The !RUN command executes the loaded segmented program.

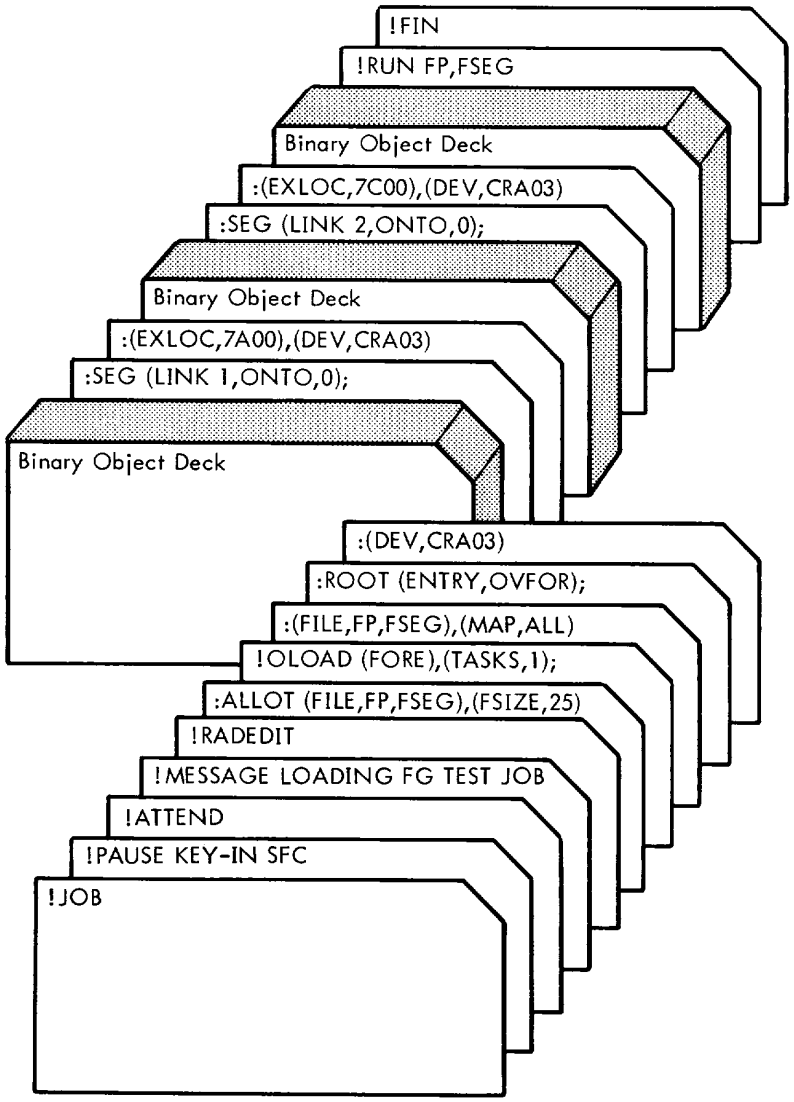
FOREGROUND JOB EXAMPLES

LOAD AND EXECUTE FOREGROUND PROGRAM



In this example, the RAD Editor allots a file, (FINT) in the Foreground Programs (FP) area of the RAD. The Overlay Loader loads the binary object deck in the file FINT in core image format. The !RUN control command causes execution of the foreground program. A PROGRAM map is specified.

LOAD AND EXECUTE SEGMENTED FOREGROUND PROGRAM



In this example, the RAD Editor allots space for a file called FSEG in the Foreground Programs (FP) area of the RAD. The Overlay Loader loads a root and two segments into FSEG in core image format. The overlaid program is executed via the !RUN control command. An ALL map is requested.

9. SYSTEM GENERATION

System Generation provides the means of forming a Monitor system adapted to the specific requirements of the user's installation. This is done by processing a set of installation control commands. The entire System Generation comprises two processes: SYSGEN and SYSLOAD. During the SYSGEN phase, only the specific installation parameters are input, not the processors. This permits the later replacement of modules on the RAD without going through an actual SYSGEN, provided that the replacements do not exceed their SYSGEN defined area. The only output from SYSGEN is an optional rebootable version of SYSLOAD (System Load).

SYSLOAD phase performs the loading of the entire RAD. That is, it loads the Monitor, the RBM Overlays, the Job Control Processor, any Optional Routines, the System Processors, User Processors, and other installation specific programs.

To eliminate the necessity for a complete reload, a new Monitor can be written without disturbing the remainder of the RAD.

SYSGEN

OVERVIEW

SYSGEN and SYSLOAD are assembled as one absolute module and then loaded by a stand alone loader. After the SYSGEN/SYSLOAD object module has been loaded, control is transferred to SYSGEN. SYSGEN inputs the installation specific parameters and sets up, in low core, all RBM tables and flags that are dependent upon these parameters. SYSGEN also builds a Symbol table containing the EBCDIC names of all RBM tables and the address where each table is loaded in memory. During the loading of RBM, this Symbol table will be used by SYSLOAD to satisfy any Monitor references (REFs) to these tables.

After SYSGEN has input its final control command, it will optionally output a rebootable binary deck in core image format containing the RBM tables, the RBM flags, and SYSLOAD. This rebootable deck can later be used to load a new version of the Monitor without going through a SYSGEN.

Upon request, SYSGEN will also output a map showing the core allocation (estimated background first word address, foreground first word address, etc.), the aforementioned Symbol table definitions and values, and the allocation of the RAD areas. The RAD area portion of the map will contain the following information:

MAP Heading	Meaning
AREA	The two-character name of the RAD area (i.e., SP, BP, FP, D3, etc.).
DISC	The device number of the RAD on which the area is located (i.e., AD0).
FWA	First word address of the area in the format xxx/yy, where xxx = track number in decimal, yy = sector number in decimal.
LWA	Last word address of the area (same format as for FWA).
NSPT	Number of sectors per track, in decimal, for the RAD on which the area is located.
NWPS	Number of words per sector, in decimal, for the RAD on which the area is located.
WP	Write protection code for the area. The codes are N No one can write in the area (unless an 'SY' key-in is in effect). B Only background can write in the area. F Only foreground can write in the area. M Only the Monitor can write in the area. X Only IOEX can write in the area.

A sample map output by SYSGEN is illustrated in Figure 17.

Control is transferred to SYSLOAD following the completion of either the SYSGEN operation or loading of the rebootable SYSLOAD deck.

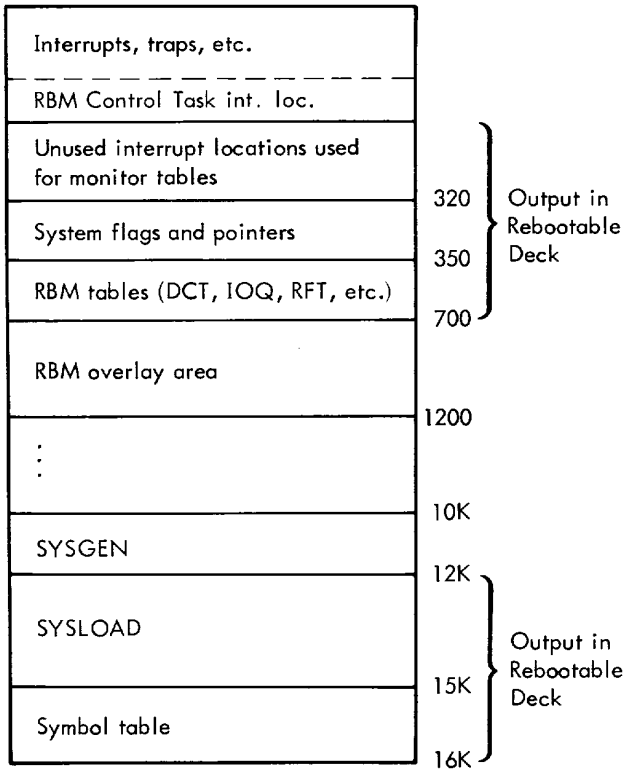
CORE ALLOCATION

CORE LAYOUT AFTER SYSGEN

After SYSGEN has executed and before control is transferred to SYSLOAD, core memory has the layout displayed below.

```
:MONITOR (CORE,32),ALLSIM,(ACCNT,F8)
:RESERVE (RSDF,4),(FFPOOL,2),(FRGD,5),(FRAD,5),(BRAD,5),(FIOQ,4),(FMBOX,100)
:DEVICE TYA01
:DEVICE CRA03
:DEVICE CPA04
:DEVICE LPA02
:DEVICE 7TAE0
:DEVICE (DCCFO,S),(ENTRACK,511),(NSPT,12),(NWPS,256),(SP,80),(FP,50);
: (BP,50),(D1,30,B),(D2,30,F);
:  (D3,5,F),(D4,5,B),(D5,5,B),(D6,5,B),(D7,5,F),(D8,5,B);
:  (D9,5,F),(DA,5,B),(DB,5,F),(DC,5,B),(DD,5,F),(DF,1,B),(XA,100)
:DEVICE 9TA80
:DEVICE 9TA81
:DEVICE 9TA82
:DEVICE 9TA83
:STDL9 (C,CRA03),(BC,TYA01),(LB,LPA02),(LL,LB),(DB,LB);
:  (BB,CPA04),(SI,CRA03),(SB,9TA81),(XX,0),(YY,0),(MT,0);
: (SI,CRA03),(BI,SI),(F1,9TA81),(F2,LB),(F3,9TA83),(F4,7TAF0);
:  (F5,0),(F6,9TA82),(CB,BB),(CI,SI)
:ALLOST (GB,8),(BV,15)
:CTINT (CT,65),(HI,65)
:INTLB (I1,60),(I2,61),(I3,62),(I4,63),(I5,64);
: (C3,5A),(FC,63),(FX,64),(IX,62)
:SYSLOAD (IN,CRA03),ALL,(V,002),(MAP,LPA02)

 BCKG. FWA=01E00
 FGD.  FWA=07800
 FMBOX FWA=07D99

**** RBM TABLE ALLOCATION ****
 MASTD=0066    DCT1=0090    DCT2=0096    DCT3=0099    DCT4=009C
 DCT5=009F    DCT6=00A2    DCT7=00A5    DCT8=00AA    DCT9=00B4
DCT10=00BF   DCT11=00C4   DCT12=00CE   DCT13=00D8   DCT14=00EE
DCT15=00F1   DCT16=00F2   DCT17=0108   DCT18=010E   DCT19=0111
 CIT1=024E    CIT2=0250    CIT3=0252    IBQ1=0254    IBQ2=0256
 IBQ3=0258    IBQ4=025A    IBQ5=025C    IBQ6=025D    IBQ7=0265
 IBQ8=0266    IBQ9=026E   IBQ10=0272   IBQ11=0274   IBQ12=0275
IBQ13=027C   IBQ14=028C    RFT1=028C    RFT2=0282    RFT3=028C
 RFT4=02C6    RFT5=02D0    RFT6=02DA    RFT7=02F4    RFT8=02E9
 RFT9=02EE   RFT10=02F3   RFT11=02FD   RFT12=0307   RFT13=0311
RFT14=0316   RFT15=031B   RFT16=0320   RFT17=0324     FP1=0336
  FP2=0342     FP3=0345     FP4=0347     FP5=034D   BPIBS1=034F
BPLBS2=035A  BPLBS3=0360  INTLB1=0366  INTLB2=036B  BVLBAD1=0370
BVLBAD2=0373 BVLBAD3=037B   WLBCK=037D  BLAYFWA=038E

**** RBM PROGRAM ALLOCATION ****
 FPSIM=06AE   DECSIM=08F6   BYTSIM=0872    CVSIM=0000    DELTA=0000
   RBM=0B34   RBMEND=1D1B

**** RAD ALLOCATION ****
 AREA    DISC     FWA      LWA     NSPT   NWPS   WP
  SP     CF0       0/ 1    79/11    12     256   N
  FP     CF0      80/ 0   129/11    12     256   N
  BP     CF0     130/ 0   179/11    12     256   N
  BT     CF0     403/ 6   511/11    12     256   B
  XA     CF0     296/ 0   395/11    12     256   X
  CK     CF0     396/ 0   403/ 5    12     256   M
  D1     CF0     180/ 0   209/11    12     256   B
  D2     CF0     210/ 0   239/11    12     256   F
  D3     CF0     240/ 0   244/11    12     256   F
  D4     CF0     245/ 0   249/11    12     256   B
  D5     CF0     250/ 0   254/11    12     256   B
  D6     CF0     255/ 0   259/11    12     256   B
  D7     CF0     260/ 0   264/11    12     256   F
  D8     CF0     265/ 0   269/11    12     256   B
  D9     CF0     270/ 0   274/11    12     256   F
  DA     CF0     275/ 0   279/11    12     256   B
  DB     CF0     280/ 0   284/11    12     256   F
  DC     CF0     285/ 0   289/11    12     256   B
  DD     CF0     290/ 0   294/11    12     256   F
  DF     CF0     295/ 0   295/11    12     256   B
**** END MAP ****
```

Figure 17. SYSGEN Map Example



RBM STRUCTURE

The RBM system is assembled in several different modules, the largest of which consists of the following nonoptional resident routines:

1. I/O Interrupt Task.
2. Control Panel Task.
3. Tasks to process the various traps.
4. The following Monitor functions: Foreground Exit, I/O Package, Interrupt Control, Segment Loader, I/O Handlers, IOEX and foreground service routines.

The other RBM parts consist of the optional resident routines, RBM Overlays, and the Job Control Processor (JCP). All RBM parts will be assembled as relocatable object modules and loaded by SYSLOAD.

The optional resident routines are the floating-point simulation routines and decimal simulation routines; they are input during SYSLOAD as required.

The RBM Overlays consist of the subtasks of the RBM Control Task, which include:

Key-in Processor

Background Abort/Exit Routine

Postmortem Dump

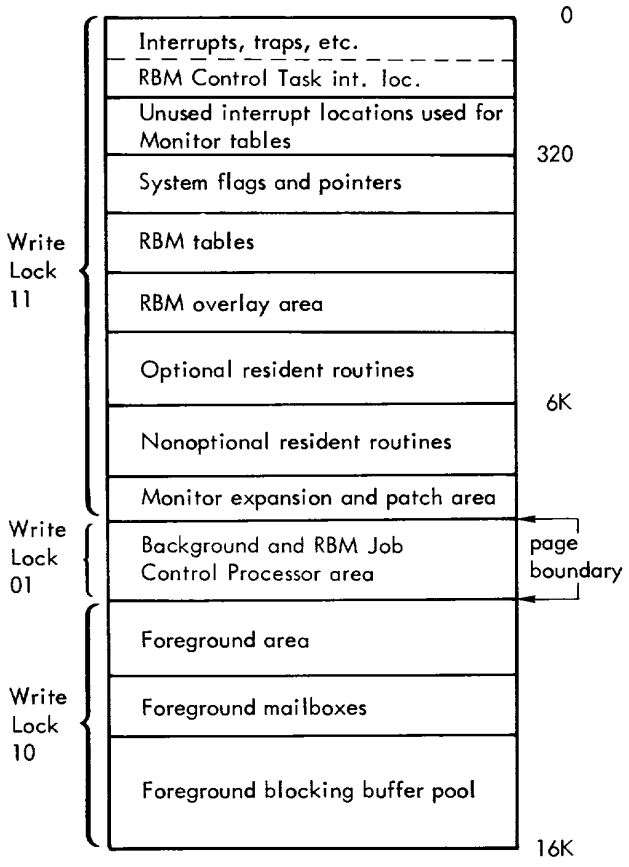
Foreground Root Loader

Background Root Loader

Checkpoint/Restart

CORE MEMORY LAYOUT AFTER SYSGEN AND SYSLOAD

After SYSGEN and SYSLOAD have executed, core memory would have the following typical layout:



Note that during SYSGEN, the user inputs the number of pages to be reserved for the foreground area. After SYSLOAD has loaded all resident routines and allocated the appropriate space for the Monitor patch area, the starting address of background will be fixed at the start of the next page. The starting address of background can not be precisely determined until all resident routines are loaded by SYSLOAD. For this reason, the background first word address output on the map by SYSGEN is necessarily an estimated address, and would be changed by SYSLOAD if the SYSGEN estimate was incorrect. The background will extend up to the start of the foreground area.

RAD ALLOCATION

RAD AREAS

During SYSGEN, the total user RAD space can be divided into a maximum of 21 areas, the size of which can not be changed except by a new SYSGEN. A subdivision of an area is a file, and each area can consist of several files. Files are defined through the RAD Editor after SYSGEN,

and can be created or deleted at any time without going through a SYSGEN process. In the order of their normal RAD allocation, the RAD areas are as follows:

AREA NAME	Name Code	Write Protect Code
System Programs	SP	N
Foreground Programs	FP	N
Background Programs	BP	N
Foreground and Background Data	D1 through DF	F or B
IOEX Access	XA	X
Checkpoint	CK	M
Background Temp	BT	B

If the RAD areas are allocated in the order given above, the user can easily protect all the programs on the RAD through the hardware write protect switches.

The user can specify a given area to physically reside on any RAD in the system if the system contains more than one RAD; however, each area must be wholly contained on one RAD. The user must also specify a RAD to be the System RAD that will contain the SP area and receive the RBM Bootstrap. The user inputs the number of words per sector and number of sectors per track for each RAD in the system and SYSGEN stores this information in the Master Directory.

The System Programs area of the RAD contains the Monitor, service processors (Overlay Loader, RAD Editor), system processors (Macro-Symbol, FORTRAN IV-H, etc.), and the System Library.

The Foreground Programs area of the RAD should contain the user's foreground programs, the Public Library, and the User Library, if they exist.

The Background Program area should contain any background programs of the user.

The IOEX Access area can be written only by IOEX and should normally be the only area of the RAD that IOEX is allowed to access.

The Foreground and Background Data areas can be used to store the appropriate type of user data. Up to fifteen Data areas (D1-DF) are allowed, to accommodate a user with multiple RADs.

The Checkpoint area is used to save the contents of background core memory during a checkpoint. The Background Temp area can be allocated to a maximum of nine scratch files (X1-X9) plus the GO and OV files.

If a user does not choose to specify the sizes for the different RAD areas, the default sizes given in Table 17 will be assumed, and the total area will be allocated to the System RAD.

Table 17. RAD Area Default Sizes

Area	Default Size	Comments
System Programs	60 tracks	Large enough to contain all system processors, one per file, in core image format; the system library in relocatable binary format; and the Monitor in core image format.
Foreground Programs	0	User is required to specify number of tracks for all areas not used by system programs.
Background Programs	0	
Foreground/ Background Data	0	
IOEX Access	0	
Checkpoint	n sectors	Where n = the initial size of background in sectors.
Background Temp	m sectors	Where m = remainder of RAD. RAD size is determined from the ENTRACK parameter on the :DEVICE command.

The areas will be physically located on the RAD in the same order as they were input during SYSGEN. The System Programs area will be the first area on the System RAD unless the user inputs the SP area in a different order. In both the initial and succeeding SYSGENs, all RAD areas required by the user must be input, except those areas that SYSGEN automatically allocates by default. In succeeding SYSGENs, the user must input all areas in the same order and with the same size as the initial SYSGEN to prevent destruction of any RAD areas.

Beginning at the starting track address input on the :DEVICE command, SYSGEN will allocate the number of tracks for each area without leaving empty spaces between areas. A bad track on a user's RAD can be skipped via an input to the RAD Editor at the time the user's files are defined. The first area allocated on the System RAD will include the RAD Bootstrap among its allocated space, and therefore, the actual space allocated for the area will be one sector less than the number of tracks input. This technique forces each area to start on a track boundary to make the hardware Write Protect switches easier to use.

BACKGROUND TEMP AREA

The scratch files (X1 through X9) of the Background Temp area of the RAD will be automatically allocated and defined

by the Job Control Processor prior to execution of a background program, unless the user wishes to override these defaults via an !ALLOBT control command. During SYSGEN, the user will not specify any standard sizes for the scratch files, X1-X9. The X1-X9 files are normally destroyed and subsequently reallocated before execution of each ~~background program in a job stack.~~ step in a job.

The GO and OV files are also in the BT area of the RAD. These files are more permanent than the X1-X9 files and are maintained throughout an entire job. The user has the option to override the default permanent size of GO and OV at SYSGEN via the :ALLOBT command. GO and OV have both a permanent size, determined at SYSGEN, and a temporary size, which can be input through the background job stack via an :ALLOBT control command.

An example of allocation for a System RAD is given in Figure 18.

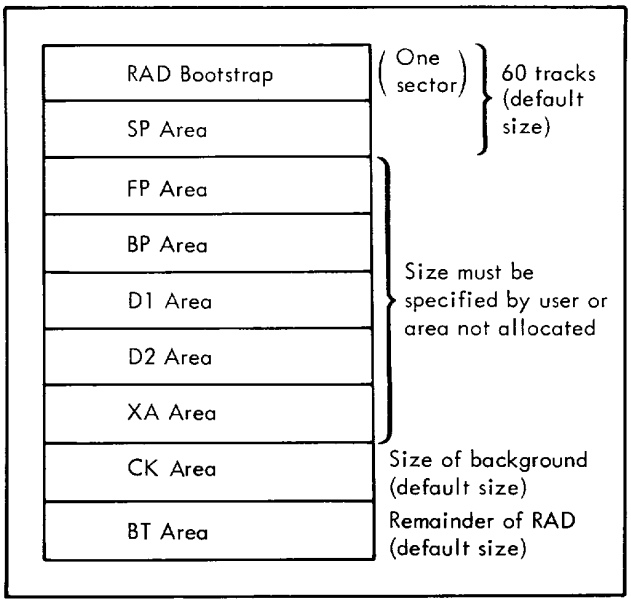


Figure 18. RAD Allocation Example

Table 18 gives the default sizes and types for GO, OV, and X1-X9, and the order in which the files are allocated. Note that X1-X9 are at the front of the BT area, and GO and OV are at the opposite end.

TABLES ALLOCATED AND SET BY SYSGEN

DEVICE CONTROL TABLE (DCT)

The DCT table is allocated by SYSGEN and several of the entries in the table are set by SYSGEN (i.e., device type, device number, dedicated to foreground bit, etc.). The DCT contains one entry for each device input by the user on the :DEVICE command, and the order of the entries is the same as the order of the :DEVICE commands. Note that there will be only one entry in the DCT for each RAD.

RAD FILE TABLE (RFT)

The RAD file table is allocated by SYSGEN from the FRAD and BRAD entries on the :RESERVE command, and should contain sufficient entries to reflect the maximum number of open RAD files that can exist simultaneously. The user will input the number of RFT entries to be reserved for foreground programs and the number to be reserved for background programs. The background is not allowed to use more than the number of RFT entries allocated for the background. However, the foreground can use all RFT entries if they are needed. The rationale for having foreground/background RAD files as opposed to a single pool of files is that a background program could erroneously use all the file entries, thus preventing the operation of a foreground program.

MASTER DIRECTORY

The Master Directory is entirely set up by SYSGEN in the resident Monitor portion of memory and contains the following information about each area on the RAD: the sector address of each area, the RAD to which the area is assigned, the sector size and number of sectors per track;

Table 18. GO, OV, X1-X9 Default Sizes

File Name	File Type	Default Size	Comments
X1 X2 X3 X4 X5 X6 X7 X8 X9	Unblocked	Determined by Job Control Processor at execution time.	File type and record sizes can be changed through a Device Mode function call or through an !ALLOBT command.
OV	Unblocked	8 tracks	Default output for Overlay Loader. Used mainly to test a program that has no permanent file defined, or to test a new version of a program without destroying the current version.
GO	Blocked (120 bytes/ logical record)	8 tracks	Used by FORTRAN and Symbol for "assemble and go" type operations.

a bit that states if an area has been allocated; and the write protection code for the area.

RBM OVLOAD TABLE

The RBM OVLOAD table is entirely set up by SYSLOAD and contains the information the Monitor needs to load a Monitor overlay. This information consists of an overlay identifier, the relative RAD address of the overlay, and the number of bytes in the overlay.

I/O QUEUE TABLE (IOQ)

The IOQ table is allocated by SYSGEN from the FIOQ and BIOQ entries on the :RESERVE command. The user inputs the maximum number of I/O operations that can be queued at one time for the foreground and background. The restrictions on the use of the foreground IOQ table are the same as for the RAD File Table.

FOREGROUND PROGRAM TABLE (FGT)

The Foreground Program table contains an active entry for each foreground program loaded into memory. Requests to load a foreground program can be made from either another foreground program or by the operator. Space for this table is allocated by SYSGEN from the FRGD entry on the :RESERVE command.

OPERATIONAL LABEL TABLE (OPLBS)

The OPLBS table is built by SYSGEN from the information input on the :STDLB command. The table has a minimum of eleven entries that contain the standard Monitor operational labels. Since operational labels are referenced via an index value in the DCB, each of the eleven standard operational labels have a fixed index value. If the user adds his own operational labels to the table, the user operational labels are assigned an index value, starting with 13, in the order in which they are input on the :STDLB command. The standard operational labels are

Op Label	Index Value	
C	1	Standard operational labels
OC	2	
LO	3	
LL	4	
DO	5	
CO	6	
BO	7	
CI	8	
SI	9	
BI	10	
SO	11	
PL	12	
xx	13	User-defined operational labels; (index value dependent upon order on :STDLB command)
yy	14	
.	.	
.	.	

INTERRUPT LABEL TABLE (INTLB)

The INTLB table is set up by SYSGEN from information contained on the :INTLB command. The table contains the name of each interrupt and the location to which the interrupt is assigned.

INPUT PARAMETERS

After the absolute object module of SYSGEN and SYSLOAD has been loaded by a stand-alone loader, control is transferred to SYSGEN.[†] SYSGEN types the following messages on the typewriter (note that the typewriter must be assigned to IOP zero, device 01; that is, TYA01):

```
RBM SYSGEN
IN, OUT DEVICES?
```

The user will input the following control command in response to the query. All SYSGEN commands must begin with a colon in column one.

```
:SYSGEN (IN,yyndd)[,(OUT,yyndd[,LP])]
```

where

IN specifies the device in the format yyndd from which the remainder of the SYSGEN control commands will be input.

yy is a device type code and must be either CR, TY, or PR (see below for a description of the codes).

n is the IOP; legal values are A-H corresponding to IOP's 0-7.

dd is the hardware device number of the device.

OUT specifies an optional output device on which the input commands are to be logged or the map, if requested, is to be output. The device type code must be either the TY or LP.

The optional LP field specifies the lower performance line printer (225 lines per minute) as opposed to the 1000 line-per-minute printer.

Following input of the :SYSGEN command, the SYSGEN control commands are input through the specified device.

[†]The stand-alone loader types out the query "INPUT DEVICE". The operator should respond by typing in the device from which the absolute object module of SYSGEN and SYSLOAD is to be loaded. Examples of a possible response are: CRA03, 9TA80, PRA05.

The following device types are standard under RBM, and should be input in the yy portion of a yyndd parameter in all SYSGEN control commands.

Device Type Code	Device
TY	Typewriter
LP†	Line printer
CR	Card reader
CP†	Card punch
9T	9-track magnetic tape
7T	7-track magnetic tape
PP	Paper tape punch
PR	Paper tape reader
DC	RAD or other disc
PL	Plotter
NO	Not a standard device. A special purpose device for use with IOEX.

SYSGEN CONTROL COMMANDS

The SYSGEN control commands are given below. The :MONITOR and :RESERVE commands (in that order) must be input prior to the :DEVICE command. A :DEVICE command must precede a :STDLB command that references that device.

:MONITOR The :MONITOR command specifies Monitor and CPU options. The :MONITOR command must precede the :RESERVE command and must precede the :DEVICE command for the System RAD.

The command has the form

```
:MONITOR (option)[,(option) . . . ,(option)]
```

where the options are

CORE,size specifies the memory size, in decimal units of K (where 1K = 1024 words), of the target computer (computer for which the SYSGEN is being run). The default value for CORE is 16K words.

FPSIM specifies that the floating-point simulation package is to be loaded by SYSLOAD. If this parameter is absent, either the floating-point hardware exists or floating-point is not needed for the target computer.

DECSIM specifies that the decimal instruction simulation package is to be loaded by SYSLOAD. The absence of this parameter indicates that either the decimal instruction hardware exists or the decimal package is not needed for the target computer.

BYTSIM specifies the byte string instruction simulation package is to be loaded by SYSLOAD.

CVSIM specifies the convert instruction simulation package is to be loaded by SYSLOAD.

ALLSIM specifies that all software instruction simulation packages are to be loaded by SYSLOAD.

ACCNT, $\left\{\begin{matrix}B\\FB\end{matrix}\right\}$ specifies that the Monitor is to perform job accounting. B or FB specifies the type of accounting. B indicates background accounting only, with all foreground time included in the background job time. FB indicates foreground/background accounting, with the foreground time kept separate from the background time. If the FB type is chosen, the foreground interrupt response time could be increased by a maximum of 5 microseconds. Absence of the ACCNT parameter indicates that no job accounting is to be kept.

LPP,value is number of lines per printer page. The default is 37. This value is used by processors that perform their own vertical format control of the printer.

:RESERVE The :RESERVE command allocates areas of core and the various variable length Monitor tables. The :RESERVE command must precede the :DEVICE command for the System RAD.

The :RESERVE command has the form

```
:RESERVE (option)[,(option) . . . ,(option)]
```

where the options are

RSDF,value specifies the decimal number of pages to be reserved for foreground programs. The value specified includes the foreground mailbox (FMBOX) and foreground blocking buffer (FFPOOL) areas, if any. This space is available for all foreground programs on a first-come, first-served basis. A program is given its predetermined core space (determined when it is loaded on the RAD by the Overlay Loader) when loaded for execution. No other program can use this space until the program is unloaded. The Public Library will also exist in this foreground space. The default value is zero.

MPATCH,size specifies the decimal number of word locations to be reserved for modifications and expansion of the Monitor. The default size is zero.

†If the optional LP (lower performance) parameter is input with a CP or LP device type, the device is the 255-line-per-minute printer in the LP case, or the 100-card-per-minute punch in the CP case (i.e., LPA02, LP or CPA04, LP).

FFPOOL,value specifies the decimal number of 256-word blocking buffers to be allocated for all foreground programs. The default value is zero.

FRGD,value specifies the maximum number of foreground programs that can reside in core memory at any one time. This parameter will be used to allocate space for the foreground program table that is used to manage the foreground core area. The default size is zero. Maximum allowable value is 225.

FRAD,value specifies the number of entries to reserve in the RAD File Table for foreground RAD files. This number should reflect the maximum number of foreground RAD files that could be open simultaneously. Note that the background RAD pool is also available to the foreground. The default value is zero.

BRAD,value specifies the number of entries to reserve in the RAD File Table for background RAD files. This number should reflect the maximum number of background RAD files that can be opened simultaneously. The default value is 5, which will be sufficient to accommodate the System Processors. The value indicated should not include the files on the BT area of the RAD.

FIOQ,value specifies the maximum number of foreground I/O operations that can be queued at any one time. This parameter determines the space allocated for foreground entries in the I/O queue table. Note that the background queue table is also available to the foreground. The default value is zero.

BIOQ,value specifies the maximum number of background I/O operations that can be queued at any one time. This parameter determines the space allocated for background entries in the I/O queue table. The default value allows three entries to be placed in the queue table.

Note that the sum of FIOQ and BIOQ must be less than 256, or an error indication will be given.

FMBOX,size specifies the decimal number of word locations to reserve at the end of the foreground core space for foreground mailboxes. The default value is zero.

BT,value specifies the maximum number of Background Temp files (X1-X9) that will ever be used. The default value is 6, that is, files X1, X2, X3, X4, X5, X6. Six files are sufficient for the System Processors. The files defined are X1-Xn, where n is the input value. The 'n' must be less than 10.

:DEVICE The :DEVICE command introduces peripheral units into the system. One :DEVICE command is required for each peripheral unit to be used. The order of the :DEVICE commands determines the Device Control Table index value that the device will receive (the index value can be used in the DCB). If an error is made in any field of the command, the entire command must be input again.

To run any foreground program, at least one foreground blocking buffer is required,

The :DEVICE command has the form

```
:DEVICE (yyndd[,LP][,S])[,(option)][,(option)]. . .
```

where

yyndd specifies the device name (see the :SYSGEN command for a description of yyndd). If yy = NO (for an IOEX device) the device will automatically be dedicated to IOEX.

LP specifies that the device is the lower performance type; e.g., LP would be used to differentiate the lower performance card punch (100 cards per minute) from the unbuffered card punch, or the lower performance printer from the high speed printer. If LP is absent, the higher performance device is assumed.

S specifies (for a RAD device only) this RAD as the System RAD; the System RAD receives the Bootstrap, the SP area, and any default allocations.

The device name must be the first field input after the :DEVICE.

The options are

DEDICATE,value specifies that the device is to be dedicated to the foreground if value is "F"; it can be used by IOEX only if value is "X". If this option is omitted, the device is undedicated unless the device is NO. In this case, the device is dedicated to IOEX.

Note: The remaining options are only applicable for a RAD device.

STTRACK,value specifies the starting track (decimal track number) on the RAD that is to be used by the system. If the option is omitted, track zero will be the starting track. Tracks are numbered starting with zero. STTRACK must be input before ENTRACK and must be equal to or less than ENTRACK.

ENTRACK,value specifies the end track (in decimal) on the RAD to be used by the system. If this option is omitted, a value of 511 will be assumed. Note that tracks are numbered 0-n, where n < 512. For a Model 7212 RAD, ENTRACK should be <64.

NSPT,value specifies the decimal number of sectors per track. The default value is 16. For a Model 7212 RAD, NSPT = 82, and for a Model 7232 RAD, NSPT = 12.

NWPS,value specifies the decimal number of words per sector. The default value is 90.

area,value specifies the decimal number of tracks to be allocated to the designated area (SP, FP, BP,

Dn, CK, XA, or BT). The various forms in which this option can be written are

SP, value[†]	XA, value
FP, value	BT, value
BP, value	Dn, value {F, B}
CK, value	where $1 \le n \le F$

If the remainder of the RAD is to be allocated to an area, "ALL" can be input instead of the number of tracks. Any area not input on a :DEVICE control command will receive its default allocation. If zero is input as a value for the number of tracks for the CK or BT area, the area will not be allocated. Note that for the data area, Dn ($1 \le n \le F$), an F (foreground), or a B (background) must be specified to indicate the write protection code for the area. See the "RAD ALLOCATION" section in this chapter.

Examples:

```
:DEVICE CRA03
```

Higher performance card reader, device number 3, on IOP number 0, undedicated.

```
:DEVICE (LPA02,LP),(DEDICATE,F)
```

Lower performance line printer, device number 2, on IOP number 0, dedicated to the foreground.

```
:DEVICE (DCB90,S),(ENT,127),(FP,15),(D1,10,F),(D2,10,B)
```

7202 RAD, device number 90, on IOP number 1, to be used as the System RAD starting on sector zero, with default sizes for the SP, CK, and BT areas, and the input sizes for the FP, D1, and D2 areas. (The BT area would receive the remainder of the RAD.)

```
:DEVICE (DCB91),(DED,X),(STTRACK,256),(NSPT,12),
    (NWPS,256),(XA, ALL)
```

7232 RAD, device number 91, on IOP number 1, to be used only by IOEX starting on track 256 and ending on track 511. These tracks are allocated to the XA area.

[†]The SP area must be allocated to the System RAD. If allocated elsewhere, an 'ERROR ITEMxx' alarm will be output.

:STDLB The :STDLB command defines all standard Monitor operational label assignments for the generated system and all standard user operational labels and their assignments. Note that operational labels cannot be assigned to RAD files during SYSGEN. The STDLB command must be input following the :DEVICE commands.

The :STDLB command has the form

```
:STDLB (label, name) [, (label, name) ...]
```

where

label specifies a standard Monitor operational label or a user operational label. All user operational labels must consist of two alphanumeric characters. Any standard Monitor operational labels not specified on a :STDLB command will receive by default a permanent assignment of zero. The order of the user's labels determines a label's position in the operational label table, and therefore determines the OPLB value that might be present in a user DCB (see the table in the example below). No label will be allowed that is the same as a Background Temp file name (GO, OV, X1-X9) or the same as a RAD area.

name specifies a physical device name to which the operational label is permanently assigned, a numeric zero, or a previously assigned operational label. In the latter case, the operational label will be assigned to the same device as the label to which it is assigned. If "0" is specified, there is no permanent assignment.

The :STDLB command example

```
:STDLB (C, TYA01), (OC, C), (LO, LPA02), (LL, LO),
    (BI, PRA05), (SI, PRA05), (PL, PLA06), (XX, PRA05),
    (ZZ, LPA02)
```

would set up the operational label table given below.

	Label	OPLB Index(10)	Permanent Assignment
Standard Monitor Op Labels	C	1	TYA01
	OC	2	TYA01
	LO	3	LPA02
	LL	4	LPA02
	DO	5	0
	CO	6	0
	BO	7	0
	CI	8	0
	SI	9	PRA05
	BI	10	PRA05
	SO	11	0
	PL	12	PLA06

	Label	OPLB Index(10)	Permanent Assignment
User Op Labels	XX	13	PRA05
	ZZ	14	LPA02

:CTINT The :CTINT command specifies the interrupt to which the RBM Control Task is to be connected, and the highest address used for interrupts on the system.

The :CTINT command has the form

```
:CTINT [(CT,address), (HI,address)]
```

where

CT,address specifies the absolute hexadecimal interrupt location to which the RBM Control Task is to be connected. If the Control Task is to be connected to an interrupt, that interrupt must be the lowest priority interrupt used by the system. If "address" has the value zero, no interrupt is available for the Control Task. In this case, the user can run only background programs, andSYSGEN will allocate the Monitor tables beginning at address X'5E'. The default value for the Control Task Interrupt is location X'61'.

HI,address specifies the highest address in hexadecimal needed for an interrupt. SYSGEN will assume that all memory locations greater than HI are unused and will attempt to allocate the Monitor tables in this area. The default value is X'13F'. Normally, CT and HI would have the same value.

:INTLB The :INTLB command provides the capability of associating a label with an interrupt location. The label may then be used in the different interrupt CALs on the Monitor.

The :INTLB command has the form

```
:INTLB (label, loc) [, (label, loc)..., (label, loc)]
```

where

label specifies a two-character alphanumeric label.

loc specifies the absolute hexadecimal interrupt location to be associated with the label.

The key-in INTLB may be used to change the assignment of the label from one interrupt location to another.

:ALLOBT The :ALLOBT command establishes the permanent sizes of the GO and OV files contained in the Background Temp area of the RAD.

The :ALLOBT command has the form

```
:ALLOBT (file name, sizes) [, (file name, size)]
```

where

file name specifies the name of the file, which must be either GO or OV.

size specifies the decimal number of tracks to be allocated for the specified file. The input size becomes the permanent size for the specified file and overrides the default sizes given in the "BACKGROUND TEMP AREA" subsection. The permanent size can be changed at execution time via an !ALLOBT control command.

:PUNCH The PUNCH command specifies that a rebootable version of SYSLOAD is to be punched after SYSGEN has input the last control command.

The PUNCH command has the form

```
:PUNCH device
```

where device specifies the device (e.g., CPA04) on which the rebootable copy of SYSLOAD is to be punched.

:SIOP The :SIOP command defines the selector IOPs as opposed to multiplexor IOPs. This command is required in SYSGEN to correctly allocate the Channel Information Table for the Monitor. All selector IOPs at an installation must be input on this command.

The :SIOP command has the form

```
:SIOP n, n, n, ...
```

where the n's indicate which IOPs are selector IOPs. The 'n' parameter must be a one-letter character from A through H.

:FIN The :FIN command signals the end of the control commands for the SYSGEN phase. Upon reading the :FIN command, SYSGEN will punch a rebootable version of SYSLOAD and output the map, if requested, and exit to SYSLOAD. The :FIN command should normally be used to terminate SYSGEN when it is not desired to continue with SYSLOAD (otherwise, the :SYSLD command should be used).

The :FIN command has the form

```
:FIN [MAP]
```

where

MAP specifies that a MAP is to be output on the same device being used to log the SYSGEN control commands. If no output device was specified on the :SYSGEN command, the MAP is output on TYA01.

:SYSLD The :SYSLD command also signals the end of the control commands to SYSGEN. The :SYSLD command causes SYSGEN to output the rebootable deck of RBM, if requested, and then exit to a SYSLOAD entry where no further control command input is required.

The :SYSLD command has the form

```
:SYSLD (IN,yyndd) [,(OUT,yyndd[,LP]), {ALL} ,
                                      {UPD}
  (V,xxxx); (MAP,yyndd[,LP])]
```

where

IN specifies the device to be used for loading the Monitor, the RBM overlays, and all optional routines. The device must be either CR, PR, 9T, or 7T. This field is not optional. See the SYSGEN command for the yyndd definition.

OUT specifies the device to receive the hard copy of the RAD Bootstrap. If the System RAD allocation starts on sector zero, this field is optional; otherwise, an output device must be specified. The output device must be either CP, PP, 9T, or 7T. The LP option specifies the lower performance card punch.

{ALL / UPD} specifies the SYSLOAD mode of operation. The "ALL" parameter indicates that all defined areas of the RAD are to be initialized to zero. The "UPD" parameter indicates that existing data on the RAD must be saved and only the new version of RBM should be output to the RAD. See "SYSLOAD", below, for a further description of these options. The default value for this parameter is "UPD".

V specifies the version number of the system being loaded. Up to four alphanumeric characters can be input for the version. The version will be logged on LL at the start of each job and logged with each postmortem dump.

MAP specifies that a MAP is to be output at the completion of SYSLOAD on the yyndd device. The device must be either LP or TY.

See the SYSGEN control command for a description of yyndd.

SYSLOAD

When the SYSGEN phase has been completed, or when the rebootable SYSLOAD deck punched by SYSGEN has been input, control is transferred to SYSLOAD. SYSLOAD loads the Monitor, the RBM Overlays and Optional Routines and outputs these to the RAD. It then outputs the RAD Bootstrap and the System Program's Directory to the RAD. When SYSLOAD terminates it enters an idle state. If necessary, the user can now load the system and user programs on the RAD by following the sequence outlined later in this chapter. If a :SYSLD command was not input to SYSGEN or input after rebooting the SYSLOAD deck, SYSLOAD will initially output the following messages on the TYA01 device:

```
RBM SYSLOAD
INPUT OPTIONS
```

The options input on the TYA01 device must be made via the :SYSLD command.

All writes made on the RAD during the SYSLOAD phase will be checked to ensure that the data was correctly recorded on the RAD.

ALL OPTION

The ALL option specifies that a complete system load is to occur and that all RAD areas should be initialized to zeros. The ALL option is necessary for the initial SYSLOAD or if the RAD allocation has changed so drastically that all areas on the RAD have moved. SYSLOAD initially zeros out all defined areas of the RAD. It then loads three groups of object modules in the following sequence: optional resident routines (FPSIM, DECSIM, CVSIM, BYTSIM); resident Monitor; and RBM Overlays and JCP.

The three groups of object modules must be loaded in the stated order, but (for example) specific RBM Overlays need not be in any special order. All these routines can be input as one package and SYSLOAD will select and load only the routines that were requested during SYSGEN, making it unnecessary to rearrange the decks of the object modules if requirements change.

Each object module is identified to SYSLOAD via a DEF item, and any object module not required is passed over. EODs are allowed between object modules, and the final object module must be followed by two EODs. If all the required object modules are not present in a group, SYSLOAD outputs the following alarm on TYA01:

```
MISSING ID name1, name2, . . .
RELOAD?
```

where $name_n$ is the name of the missing routine, the name being indicated by the only DEF item in the object module.

If "Y" (YES) is input to the RELOAD query, SYSLOAD again reads the input device to load the missing routines. If "N" (NO) is input, SYSLOAD assumes the missing routines are not required and continues.

SYSLOAD writes the required overlays on the RAD as they are loaded and sets up the information needed to load the overlays in the RBM OVLOAD table. Modules that are not overlays will be loaded directly into core and later written out with resident RBM.

When the directory is written on the RAD by SYSLOAD, the System Programs Directory will contain entries for two files named "RBM" and "RADBOOT". RADBOOT is the file that contains a copy of the RAD Bootstrap, which is the only program on the RAD not contained within a RAD area. Therefore, it cannot be accessed if a RAD dump or save is required, and for this reason a copy of the Bootstrap is kept in the SP area.

The RBM and RADBOOT files will be the first two files in the SP area. The area on the RAD allocated for the Monitor will include any patch or expansion core area the user has requested. If a new version of the RBM system exceeds the RAD space allocated to a previous version, all programs in the SP area must be reloaded.

After RBM and the SP Directory are output to the RAD, SYSLOAD sets up the appropriate command list in the Bootstrap to enable the Bootstrap to easily reload RBM from the RAD. The Bootstrap is then written both into the RADBOOT file and onto the starting sector of the System RAD (SSTRACK option on :DEVICE command).

If the System RAD allocation starts at other than sector zero, a copy of the RAD Bootstrap is punched on the device specified by the OUT keyword on the :SYSLD command. The user can then boot in RBM by loading the hard copy of the RAD Bootstrap. This permits having more than one Monitor system on the user's RAD and still being able to boot in RBM by reading in a hard copy of the Bootstrap.

UPDATE OPTION (UPD)

The UPD option can be used whenever there is an existing version of RBM on the RAD, and the user wishes to load a new version of the Monitor or change some of the SYSGEN parameters. It is not necessary to go through a SYSGEN to load a new version of the Monitor. It is only necessary to load the rebootable SYSLOAD deck and go through a normal SYSLOAD, specifying the "UPD" option on the :SYSLD command.

To change any SYSGEN-defined parameters, it is necessary to input the complete set of SYSGEN control commands. That is, there is no attempt to merge the new version of the Monitor with the existing version on the RAD.

If the user does not want to disturb any of the RAD areas, the RAD areas must be input with the same size and in the same order as the initial SYSGEN. If the size of a RAD area has to be changed or a new RAD area has to be added, all RAD areas (except CK or BT) must be reloaded from the first changed area to the end of the RAD. Therefore, the areas most subject to change in size should be allocated to the end of the RAD so that the minimum number of areas are affected by a change. An area that must be moved can be saved and restored intact by using the RAD Editor Save and Restore functions. It is normally to the user's advantage to take the default size and allocation for the CK and BT areas since these are automatically allocated at the end of the RAD and may be changed without affecting any other area.

To inform the user as to which areas on the RAD have moved, SYSLOAD reads in the RAD Bootstrap from the existing version, determines where the Monitor is located on the RAD, and then inputs the Master Directory from the existing version. If the absolute RAD first word address is changed for any of the SP, FP, BP, XA, or Dn areas, SYSLOAD outputs an appropriate alarm, requests permission to continue, and then zeros out the first sector in each area that has moved, thus effectively erasing all data in the area. The alarms that could be output are

RELOAD
SP AREA
FP AREA
BP AREA
Dn AREA (where $1 \leq n \leq F$)
XA AREA
CONTINUE?

If the user types "YES" to the CONTINUE? query, SYSLOAD will proceed and effectively erase each area that has moved. A "NO" input is allowed in the event that the user made an error in allocating the RAD areas on the :DEVICE command and does not wish to proceed. For a "NO" input, SYSLOAD will output the map, if requested, and then enter a "WAIT" condition. Note that since SYSLOAD must read in the RAD Bootstrap of the existing version to find the RAD location of the Master Directory, the starting track of the System RAD must be identical in both versions for the "UPD" option to be used.

The RELOAD, SP AREA alarm would also be output if the new version of the Monitor occupied more or less RAD space than the existing version. Since the Monitor is the first file in the area, all other files have to be moved and reloaded if the new Monitor requires a different amount of RAD space. In this case, the user must reload the entire SP area in the same manner as in an initial system load. The Monitor normally does not overflow its allocated RAD space when a new version is loaded, since RAD space is allocated up to the starting address of background.

If the first word address of background is different in a new version from that of the existing version, the alarm

RELOAD, BGKG, PROGRAMS

is output. All programs that execute in the background, both System Processors and user background programs, would then have to be reloaded and absolutized for their new core execution location.

If SYSLOAD determines that the new version is completely compatible with the existing version, the message

RELOAD, NOTHING

is output.

After typing the necessary RELOAD alarms, SYSLOAD loads the resident optional routines, the resident Monitor, and the RBM Overlays as described under the ALL option.

RAD ALLOCATION OF SP AREA

When SYSLOAD has executed, the Systems Program area of the RAD will have the following layout:

SP Directory Entries for RBM, BOOT	Relative sector 0
RBM File (Resident Monitor and RBM Overlays)	Sectors 1 - n
BOOT File	Sector n + 1
Unused SP AREA	Sectors (n + 2) - (n + m)

SYSGEN AND SYSLOAD ALARMS

All alarm messages that can be output during SYSGEN and SYSLOAD are defined in Table 19.

LOADING SYSTEM PROCESSORS AND USER PROGRAMS

After SYSLOAD completes its operation it will type the message

SIGMA 5/7 RBM-2 VERSION XXXX

and executes a WAIT instruction. The operator should then place his job stack in the C device to load the appropriate programs, perform an interrupt, and key in a "C". Control will be transferred to the Job Control Processor to read the first control command.

If the "ALL" option was input to SYSLOAD, or if the SP AREA or BCKG PROGRAMS need reloading, the RAD Editor must be the first processor loaded. The RAD Editor should be loaded by the JCP Loader onto the OV file. The user then inputs control commands to the RAD Editor to define permanent RAD files for all system processors (including the RAD Editor), libraries, and all user programs. The RAD Editor can then be copied from the OV file onto its permanent file via the RAD Editor COPY command. The Overlay Loader should be the next processor loaded by the RBM Loader onto its defined file, and this loader can then be used to load all other processors and user programs.

Typical examples for loading the RAD Editor and Overlay Loader, followed by a load of a system processor are illustrated in Figures 19 and 20.

Table 19. SYSGEN and SYSLOAD Alarm Messages

Alarm	Meaning	Recovery Action
	Non-I/O Alarms	
BI CKSM ERR	A checksum error has occurred in the object module being input.	SYSLOAD will execute a "WAIT" instruction. If the computer is put back into RUN, the next record will be read from the BI input device.
BI SEQ ERR	A sequence error has occurred in the object module being input.	Identical to "BI CKSM ERR".
BT AREA TOO SMALL	The space allocated to the BT area of the RAD is insufficient to hold the default sizes of the GO and OV files.	There is no recovery from this condition except to rerun the SYSGEN to either allocate more RAD space for the BT area, or reduce the default size of the GO and/or OV file.
CK AREA TOO SMALL	The amount of RAD space allocated for the CK area is not sufficient to hold the initial size of background.	Either the RAD areas must be reallocated (requires a rerun of SYSGEN) or a checkpoint cannot be done with the initial size of background.
DUP. DEF,xxxxxxxx	The same DEF has been encountered in two object modules, probably indicating that two copies exist of the same object module.	Identical to "ERR, CONTROL BYTE = xx".
EOF BEFORE END ITEM	During the loading of an object module, SYSLOAD has encountered a misplaced EOD or EOF.	SYSLOAD will enter a "WAIT" condition. If the computer is put back into RUN, the EOD or EOF will be ignored and the next record will be input.

Table 19. SYSGEN and SYSLOAD Alarm Messages (cont.)

Alarm	Meaning	Recovery Action
	Non-I/O Alarms (cont.)	
ERR, CONTROL BYTE =xx	The xx control byte in the object module being loaded cannot be processed by SYSLOAD.	SYSLOAD will execute a "WAIT" instruction. If the computer is put back into RUN, the current object module will be bypassed and not loaded.
ERROR ITEM xx	An error has occurred in item xx of the last control command input. Every item (except the :), followed by a blank or a comma, is counted in determining the one in error. If xx is one greater than the last item input, a nonoptional item was not input.	Control will be transferred to TYA01 to allow the user to correct the error. Unless stated otherwise (where the individual commands are described) all items preceding the incorrect one have been processed, and only items starting with and following the incorrect one need be input. If the user desires to input nothing from TYA01 and to transfer control back to the original input device, a single colon (:) should be input on TYA01. If an error occurs on a continuation card, a card containing a control command must follow.
ILL. DEF, xxxxxxxx ILL. REF, xxxxxxxx	The specified DEF or REF is not recognized by SYSLOAD during the loading of an object module.	SYSLOAD will enter a "WAIT" condition. If the computer is put back into RUN, processing of the current object module will continue.
INPUT ORDER ERROR	The :MONITOR, :RESERVE, or :DEVICE command for the System RAD has been input in the wrong order.	Catastrophic error. Rerun SYSGEN from the start.
MISSING ID	See ALL option.	See ALL option.
NO SYSTEM RAD	No RAD has been designated as the System RAD.	Catastrophic error. SYSGEN must be rerun from the start.
OBJ. MOD. NOT RECOG.	The current object module being loaded by SYSLOAD is not recognized by SYSLOAD.	Identical to "ERR, CONTROL BYTE = xx".
OC LABEL NOT ASSIGNED	The OC operational label has not been assigned to a typewriter device.	There is no recovery from this error. The OC label must be assigned in order for the system to function.
RAD OVERFLOW	The total number of tracks input on the :DEVICE command have exceeded the total available size.	The :DEVICE command must be completely reinput, with the sizes of the areas appropriately changed.
RELOAD SP AREA FP AREA BP AREA Dn AREA XA AREA BCKG. PROGRAMS NOTHING	See UPDATE option.	See UPDATE option.
TYPE C - OR - E	This message is output after each object module when RBM is being loaded from paper tape to allow the user to load a new paper tape for each object module.	Type "C" to continue if this is not the last object module to be input; or "E" (meaning EOD) if this is the last object module.

Table 19. SYSGEN and SYSLOAD Alarm Messages (cont.)

Alarm	Meaning	Recovery Action
	Non-I/O Alarms (cont.)	
UNABLE TO FIND OLD RBM	During an update run, SYSLOAD was unable to locate the old version of RBM on the RAD.	SYSLOAD will continue with the load, but will be unable to make any checks as to which areas need reloading. The user must reload the entire SP area of the RAD if this alarm is output.
	I/O Alarms	
yyndd BUSY IOP n BUSY	The indicated device or IOP has returned a busy status.	SYSGEN will keep attempting the I/O operation. Probably indicates a hardware problem.
yyndd ERROR, SB = xxxx yyndd PARITY, TRK = xxxx	A transmission error has occurred with the indicated device. SB = xxxx indicates the contents of the TIO status bytes in hexadecimal. If a parity occurs while clearing the RAD, the bad track, as returned in the sense order, is also logged in hexadecimal.	SYSGEN continues attempting the I/O operation, unless a parity has occurred while clearing the RAD. In this case, this alarm and the parity alarm will be logged and the RAD clearing will continue.
yyndd FAULT, TDV = xxxx	A hardware fault has occurred on the indicated device. The TDV status byte is also output in hexadecimal.	SYSGEN continues attempting the I/O operation. Repair and ready the indicated device.
yyndd MANUAL	The indicated device is in manual mode.	Ready the device.
yyndd UNRECOG	The device indicated by yyndd is unrecognized by the system.	SYSGEN will enter a "WAIT" state. Probably an invalid device number was input, and the SYSGEN will have to be rerun from the start. If the "WAIT" state is cleared, SYSGEN will retry the I/O operation.
yyndd UNUS. END, TDV = xxxx	An unusual end status has been returned from the indicated device. The TDV status byte is also logged in hexadecimal.	SYSGEN continues attempting the I/O operation.
yyndd WRT PROT	The indicated magnetic tape or RAD is hardware write-protected.	For a magnetic tape, insert a write ring and ready the tape. For a RAD, reset the hardware Write Protect switch and then clear the "WAIT" state so SYSLOAD can retry the I/O operation.

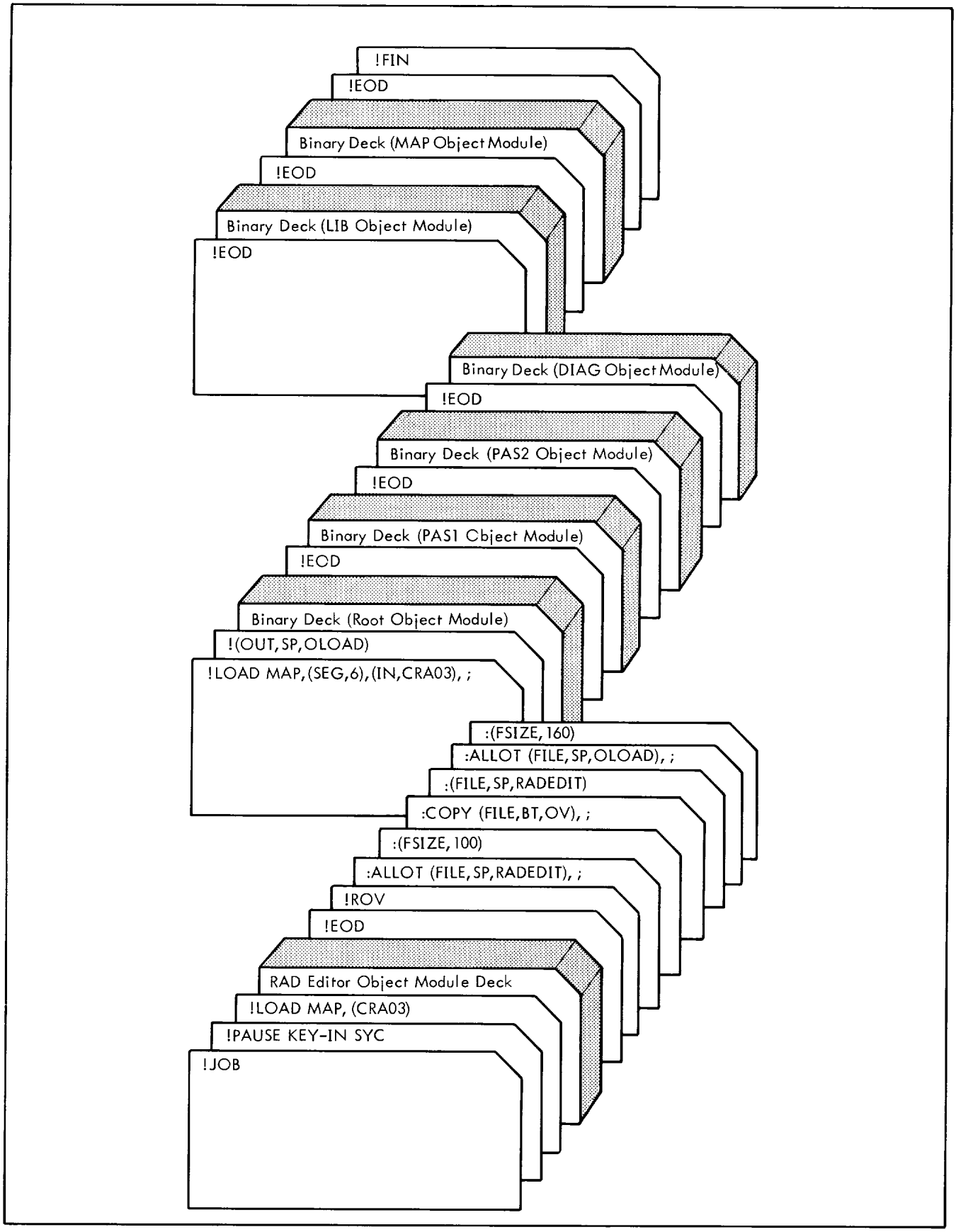


Figure 19. Loading RAD Editor and Overlay Loader Processors Into System

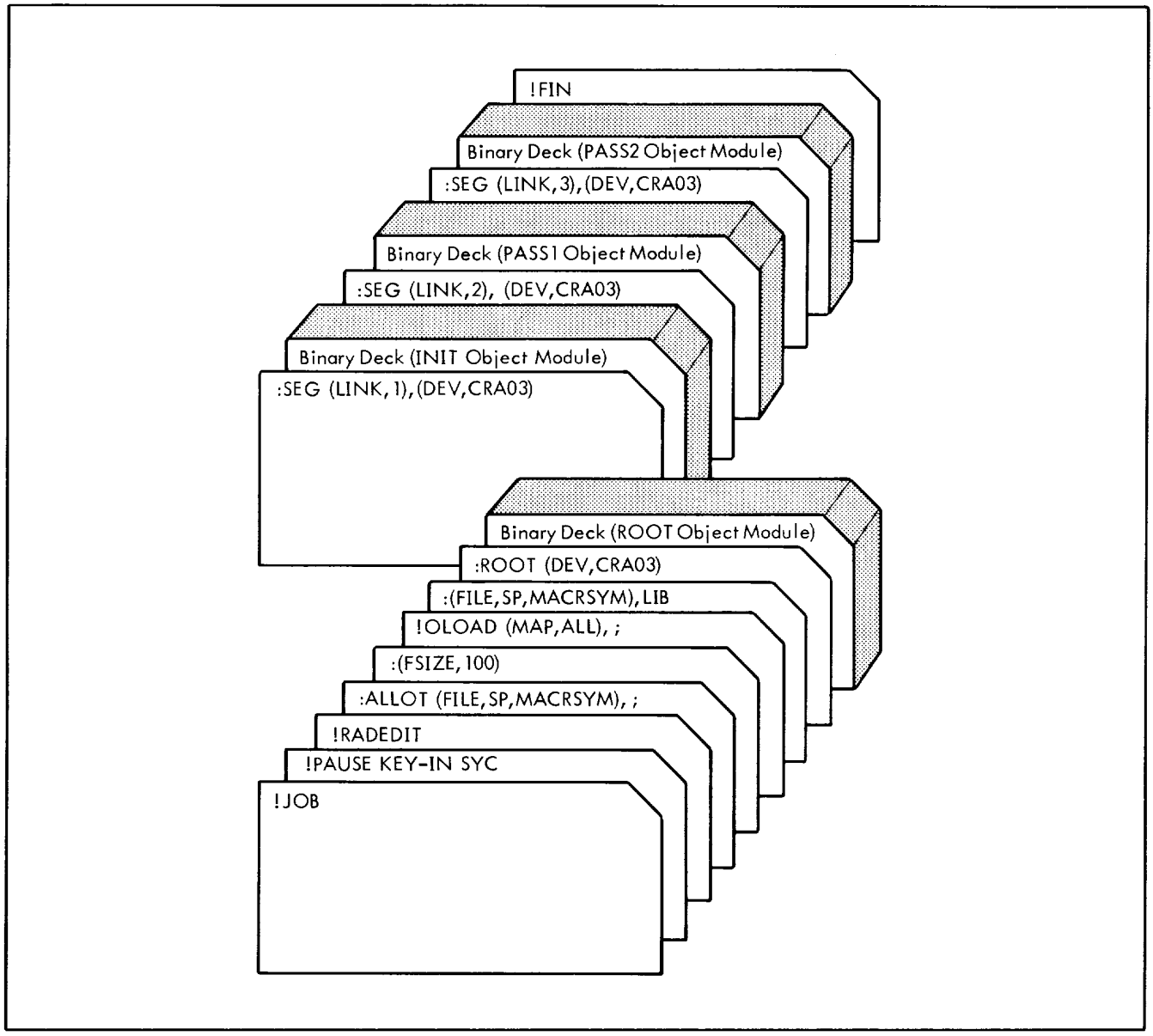


Figure 20. Loading Macro-Symbol Processor Into System

APPENDIX A. SIGMA STANDARD OBJECT LANGUAGE

INTRODUCTION

GENERAL

The XDS Sigma standard object language provides a means of expressing the output of any Sigma 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 XDS Sigma 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 XDS 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 XDS 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    AI,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.

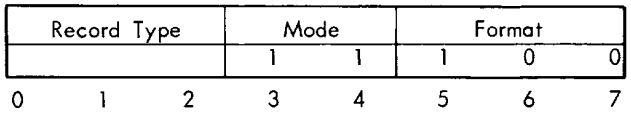
The ordered set of logical records that a processor generates for a program or subprogram is termed an "object module". The end of an object module is indicated by a module-end type code followed by the error severity level assigned to the module by the processor.

RECORD CONTROL INFORMATION

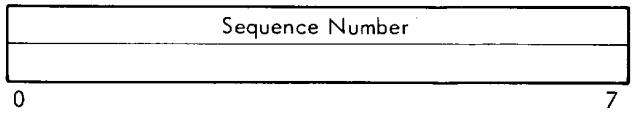
Each record of an object module consists of 4 bytes of control information followed by a maximum of 104 bytes of load information. That is, each record, with the possible exception of the end record, normally consists of 108 bytes of information (i.e., 72 card columns).

The 4 bytes of control information for each record have the form and sequence shown below.

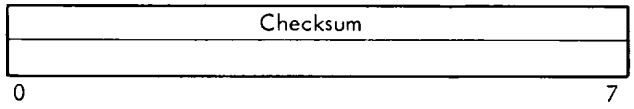
Byte 0



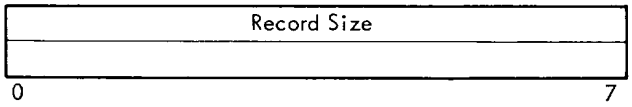
Byte 1



Byte 2



Byte 3



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 ($5 \leq$ record

size ≤ 108). The record size 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 XDS 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 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 Nonstandard 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 XDS 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

†If the module has fewer than 256 previously assigned name numbers, this byte is absent.

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

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.

[†]If the module has fewer than 256 previously assigned name numbers, this byte is absent.

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

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

†If the module has fewer than 256 previously assigned name numbers, this byte is absent.

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 ($1 \leq L \leq 255$) and terminates in bit position T, where:

T = current load bit position -256 +K.

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

CONTROL BYTES (In Binary)

Begin Record	Record number: 0		
00111100	Record type: not last, Mode binary, Format: object language.		Record control information not part of load item
00000000	Sequence number 0		
01100011	Checksum: 99		
01101100	Record size: 108		
	03020101 (hexadecimal code comprising the load item)		Source Line 1
00000011	Declare external definition name (2 bytes) Name: AA	Declaration number: 1	
	03020202		
00000011	Declare external definition name (2 bytes) Name: BB	Declaration number: 2	
	03020303		
00000011	Declare external definition name (2 bytes) Name: CC	Declaration number: 3	
	0502D9E9		Source Line 2
00000101	Declare primary reference name (2 bytes) Name RZ	Declaration number: 4	
	0503D9E3D5		
00000101	Declare primary reference name (3 butes) Name: RTN	Declaration number: 5	
	0A0101000003202000002		Source Line 5[†]
00001010	Define external definition		
	Number 1		
00000001	Add constant: 800 X'320'		
00100000	Add value of declaration (byte resolution)		
	Number 0		
00000010	Expression end		
	040100000320200002		Source Line 4
00000100	Origin		
00000001	Add constant: 800 X'320'		
00100000	Add value of declaration (byte resolution)		
	Number 0		
00000010	Expression end		
	4422000000		Source Line 5
01000100	Load absolute the following 4 bytes: X'22000000'		
	07EB0426000002		
00000111	Define field		
	Field location constant: 235 bits		
	Field length: 4 bits		
	Add the following expression to the above field:		
00100110	Add value of forward reference (word resolution)		
	Number 0		
00000010	Expression end		

[†]No object code is generated for source lines 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.

	8432000000	
10000100	Load relocatable (short form). Relocate address field (word resolution) Relative to declaration number 4 The following 4 bytes: X'32000000'	
	07EB0426000602	
00000111	Define field Field location constant: 235 bits Field length: 4 bits Add the following expression to the above field:	Source Line 6
00100110	Add value of forward reference (word resolution) Number 6	
00000010	Expression end	
	CC50000000	
11001100	Load relocatable (short form). Relocate address field (word resolution) Relative to forward reference number 12 The following 4 bytes: X'50000000'	
	07EB0426000602	
00000111	Define field Field location constant: 235 bits Field length: 4 bits Add the following expression to the above field:	Source Line 9
00100110	Add value of forward reference (word resolution) Number 6	
00000010	Expression end	
	D269200000	
11010010	Load relocatable (short form). Relocate address field (word resolution) Relative to forward reference number 18 The following 4 bytes: X'69200000'	Source Line 11
	4420000001	
01000100	Load absolute the following 4 bytes: X'20000001'	
	07EB0426000002	
00000111	Define field Field location constant: 235 bits Field length: 4 bits Add the following expression to the above field:	Source Line 13
00100110	Add value of forward reference (word resolution) Number 0	
00000010	Expression end	
	80680000CA	
10000000	Load relocatable (short form). Relocate address field (word resolution) Relative to declaration number 0 The following 4 bytes: X'680000CA'	Source Line 14
	8568000000	
10000101	Load relocatable (short form). Relocate address field (word resolution) Relative to declaration number 5 The following 4 bytes: X'68000000'	Source Line 15
	08	
00001000	Define forward reference (continued in record 1)	Source Line 16

Begin Record	Record number 1	
00011100	Record type: last, Mode: binary, Format: object language.	Record Control Information
00000001	Sequence number 1	
11101100	Checksum: 236	
01010001	Record size: 81	
	000C010000033C200002 (continued from record 0)	Source Line 16
	Number 12	
00000001	Add constant: 828 X'33C'	
00100000	Add value of declaration (byte resolution)	
	Number 0	
00000010	Expression end	
	42001	
01000010	Load absolute the following 2 bytes: X'0001'	
	080006010000000302	Source Line 17
00001000	Define forward reference	
	Number 6	
00000001	Add constant: 3 X'3'	
00000010	Expression end	
	080000010000000402	Source Line 18
00001000	Define forward reference	
	Number 0	
00000001	Add constant: 4 X'4'	
00000010	Expression end	
	0F00024100	Advance to Word Boundary
00001111	Repeat load	
	Repeat count: 2	
01000001	Load absolute the following 1 bytes: X'00'	
	0800120100000340200002	Source Line 19
00001000	Define forward reference	
	Number 18	
00000001	Add constant: 832 X'340'	
	Add value of declaration (byte resolution)	
	Number 0	
00000010	Expression end	
	0A020100000340200002	
00001010	Define external definition	
	Number 2	
00000001	Add constant: 832 X'340'	
00100000	Add value of declaration (byte resolution)	
	Number 0	
00000010	Expression end	
	44224FFFFF	
01000100	Load absolute the following 4 bytes: X'224FFFFF'	
	0D0100000320200002	Source Line 22
00001101	Define start	
00000001	Add constant: 800 X'320'	
00100000	Add value of declaration (byte resolution)	
	Number 0	
00000010	Expression end	

```
            0B000344
00001011    Declare standard control section declaration number: 0
            Access code: Full access.  Size 836  X'344'

            0E00
00001110    Module end
                Severity level: X'0'
```

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. REAL-TIME PERFORMANCE DATA

RESPONSE TO INTERRUPTS BY CENTRALLY CONNECTED TASKS

Table B-1 shows the time (in μsec) used by the system to save the interrupted context and establish the interrupting task context. This time includes the XPSD in the interrupt location context and represents the total time between the interrupt becoming active and the start of execution of the first instruction in the task (assuming no interruption by a higher priority task). The minimum and maximum times reflect minimum and maximum memory overlap.

I/O INTERRUPT

Following successful completion of an I/O device access, the I/O interrupt will remain active for a maximum of 90 μsec, assuming that no higher priority interrupts have become active during this period. RBM never disables the interrupt system for longer than 100 μsec.

Upon clearing the I/O interrupt, the system proceeds with cleanup of the request at the priority level of the interrupted task.

CONSOLE INTERRUPT

The Console Interrupt remains active for less than 30 μsec. During this time, a flag in the Control Task is set to indicate the occurrence of the interrupt and the Control Task interrupt is triggered.

OVERLAY LOADING

Overlay loading is accomplished with a negligible percentage of total time devoted to non-I/O system activity. For example, on the Model 7202/7204 RAD, a 1400-word overlay requires approximately 50 msec, assuming average latency (17 mils) and I/O transfer time of 34 msec. To this must be added the time waited to gain access to the RAD (time of request, if any in progress, plus the time of any higher priority requests).

Table B-1. Times Required to Save Interrupted Context

Registers Saved	Times in μsec[t]	Task interrupts background with accounting	Task interrupts system or foreground tasks with accounting	No accounting
4	Min.	39.0	34.4	30.8
	Max.	42.2	36.1	32.4
8	Min.	42.2	37.6	34.0
	Max.	44.8	39.7	36.0
16	Min.	48.6	44.0	40.4
	Max.	52.0	46.9	43.2

[t]Times assume a Sigma 7 configuration without map.

APPENDIX C. RAD STORAGE REQUIREMENTS

Assuming a Model 7202/7204 RAD, a rough approximation of the amount of RAD space required by the RBM system is 60 tracks to store the Monitor plus the following processors:

Macro-Symbol

FORTRAN IV-H

FORTRAN Math/Run-Time Library

Symbol

Overlay Loader

RAD Editor

The RAD areas used by the system require the following approximate numbers of tracks:

RAD Area	Number of Tracks
System Programs	50
Checkpoint	8 (assuming an 11K background)
Background Temp[†]	45 (sufficient for a Macro-Symbol "assemble and go" of about a 7000 line source program)

[†]The size of the Background Temp area is highly dependent on a user's requirements.

APPENDIX D. JCP LOADER

In addition to loading the Overlay Loader and RAD Editor into their respective files in the SP RAD area at SYSGEN time (see !LOAD command in Chapter 2), the JCP Loader can be used to load some user programs. Within the restrictions given below, both nonoverlaid and overlaid programs can be loaded onto the RAD for subsequent execution.

LOADING NON-OVERLAID PROGRAMS

1. The last object module in the program being loaded must be terminated by an !EOD command. Any number of object modules can be loaded.

2. The object module cannot contain any of the following load items: declare dummy section, declare secondary external reference name, forward reference definition, or add or subtract absolute section. The field (in a define field load item) cannot cross a word boundary.

3. FORTRAN compiled programs or programs assembled with basic Symbol, or programs using the LOCAL directive cannot be loaded with this Loader.

4. Object modules being loaded cannot contain more than 225 declaration name numbers.

5. The JCP Loader creates the OVLOAD and DCB table and the M:SL DCB. The user must code the complete PCB except for the OVLOAD table address, the DCB table address, and the M:SL DCB address. The user must also code all DCBs (except M:SL), the Temp Stack and all other tables referenced in the PCB.

6. The PCB must be the first data loaded in the root.

Example:

Load a nonoverlaid foreground program onto its permanent file:

```
!LOAD (IN,9TA82),(OUT,FP,FCOPY),(EXLOC,4100),F
```

This example loads a foreground program (indicated by F parameter) from a 9-track tape onto the FCOPY file in the FP area. The program will be loaded to execute at 4100_{16} (EXLOC,4100). The foreground program can consist of any number of object modules, but the last object module must be followed by an EOF. The object modules must obey the restrictions specified for the JCP Loader (see above). Note that an SY key-in must be in effect.

LOADING OVERLAID PROGRAMS

For loading a program with overlay segments, the above restrictions plus the following restrictions apply:

1. The only overlay structure allowed is a root plus one level of overlay.

2. The root must be the first group of object modules loaded and must be terminated by an !EOD command. Each group of object modules loaded after the root and terminated with an !EOD command will consist of one overlay segment and will be attached to the end of the root. Each overlay segment will be assigned a segment identification of 1 to n, where 1 is assigned to the first segment loaded, 2 assigned to the second segment loaded, etc. The segment identification is used in calling in an overlay segment at execution time.

3. The number of overlay segments must be correctly stated on the !LOAD command.

For multisegment programs, the JCP Loader builds the OVLOAD table at the end of the root and allocates seven words for the M:SL DCB. The entry address for each overlay is taken from the last encountered start item and placed in the OVLOAD table along with the load address, byte count, and segment identification.

The JCP Loader writes the program in core image format onto the appropriate file. The addresses and names of all M: or F: DEFs will be used by the Loader to create the DCB table. The loaded program can be executed via the RUN, ROV, or Name command, depending upon which RAD file the program was loaded on. Note that the Loader never loads a program directly into core memory. The JCP Loader can be used to load foreground programs (which must obey the previously stated restrictions) by using the EXLOC and F parameters on the !LOAD command.

APPENDIX E. SYSTEM PATCHING

Modification of the resident RBM system (including all system tables), RBM overlays, and Job Control Processor can be performed at system boot time through patches defined on !MODIFY control commands. Any number of !MODIFY commands can be input, and the stack is terminated by a single !END command. The system is notified that patching is to take place by sense switch settings that are set prior to RBM initialization.

As each command is processed, the overlay is read into core, modified, and written out to the system RAD. Note that unused RAD space between the end of an overlay and the end of an overlay's file (that is, unused space on the last sector of an overlay) can be used for a patch area.

INPUT OPTIONS

The !MODIFY control commands are input at system boot time during the RBM initialization. Sense switches 1, 2, and 3 are used to designate the input device as follows:

Setting	Meaning
All off	no patches
1 on; 2, 3 off	input from C device
2 on; 1, 3 off	input from OC device
3 on; 1, 2 off	input from SI device

Any other sense switch settings will result in an error message and the system will have to be rebooted. The switches must be set prior to booting in the system.

COMMAND FORMATS

In the !MODIFY control commands listed below, the brackets only indicate options available in the specification field and are not actually used in the MODIFY statements; the indicated parenthesis and commas are required. The general format for !MODIFY commands is

```
!MODIFY (module,loc), value[R]
```

where

module specifies one of the modules described below to be patched.

loc specifies the initial hexadecimal location to be modified and is relative to the beginning of the named module (for ABS, loc is relative location 0).

value specifies the hexadecimal value to be inserted in the location. Successive values in successive locations.

R is the relocation flag indicating that the address field of the value is to be relocated by adding the beginning address of the named module.

PATCH SYSTEM OVERLAY OR JCP

```
!MODIFY (OLAY,name,loc)[R]value[,[R]value,... [R]value]
```

where name can be one of the following:

JCP	(Job Control Processor)
CKPT	(Checkpoint Routine)
FGL1	(Foreground Program Releaser)
FGL2	(Foreground Program Loader)
ABEX	(Abort/Exit Routine)
KEY1	(Key-in Processor 1)
KEY2	(Key-in Processor 2)
PMD	(Postmortem Dump Routine)
BKL1	(Background Loader 1)
BKL2	(Background Loader 2)

PATCH SIMULATION ROUTINE

```
!MODIFY (SROU, name,loc) [R]value[,[R]value... [R]value]
```

PATCH RBM MONITOR

```
!MODIFY (RBM, loc), [R]value[, [R]value ... [R]value]
```

PATCH SYSTEM TABLES

```
!MODIFY (ABS, loc), value[, value, ... value]
```

PATCH PATCH AREA

```
!MODIFY (PATCH, loc), value
```

!END COMMAND FORMAT

The !MODIFY control command(s) must be followed by a terminating !END command with the form

```
!END [RBM]
```

where RBM indicates that the copy of RBM resident on the system RAD is to be replaced by the current version of RBM in core.

SYSTEM PATCHING DIAGNOSTICS

The diagnostic messages in Table E-1 will be output on the OC device for a corresponding error in sense switch settings or !MODIFY commands. There is no error recovery. In all cases, the condition causing the message must be corrected and the system rebooted.

Table E-1. System Patching Diagnostics

Message	Meaning	Action
!!INVALID SSW SETTING, REBOOT	Switch 4 is set, or more than one switch is set.	Correct and reboot system.
!!ERROR ITEM xxxx, REBOOT	Illegal area, illegal simulation routine, illegal identifier, or missing syntax (e.g., parenthesis or comma).	Correct and reboot system.
!!PATCHING OUTSIDE OF SPECIFIED AREA, REBOOT	Requested patch either exceeds or falls outside previously specified area.	Correct and reboot system.
!!PATCH AREA NOT ALLOCATED	Patch area was not previously specified at SYSGEN time.	Re-SYSGEN and reboot system.

INDEX

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

G

H

I

J

K

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

L

M

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

S

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

T

U

V

W

X