Xerox Real-Time Batch Monitor (RBM)

Xerox 530 and Sigma 2/3 Computers

Real-Time and Batch Processing
Reference Manual

90 10 37I

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox Real-Time Batch Monitor (RBM)

Xerox 530 and Sigma 2/3 Computers

Real-Time and Batch Processing
Reference Manual

90 10 371

February 1975

Price: $6.50

Printed in U.S.A.

REVISION

This publication is a major revision of the Xerox Real-Time Batch Monitor (RBM)/RT, BP Reference Manual for Xerox 530 and Sigma 2/3 computers, Publication Number 90 10 37H (dated June, 1973). Technical changes made to the text are for the G00 version of RBM. All technical changes from that of the previous manual are indicated by a vertical line in the margin of the page.

RELATED PUBLICATIONS

Title	Publication No.
Xerox 530 Computer/Reference Manual	90 19 60
Xerox Sigma 2 Computer/Reference Manual	90 09 64
Xerox Sigma 3 Computer/Reference Manual	90 15 92
Xerox Real-Time Batch Monitor (RBM)/OPS Reference Manual	90 15 55
Xerox Real-Time Batch Monitor (RBM)/SM Reference Manual	90 30 36
Xerox Real-Time Batch Monitor (RBM)/User's Guide	90 17 85
Xerox Real-Time Batch Monitor (RBM)/System Technical Manual	90 11 53
Xerox Extended Symbol/LN,OPS Reference Manual	90 10 52
Xerox Symbol/LN,OPS Reference Manual	90 10 51
Xerox Basic FORTRAN and Basic FORTRAN IV/LN Reference Manual	90 09 67
Xerox Basic FORTRAN/OPS Reference Manual	90 10 61
Xerox Basic FORTRAN IV/OPS Reference Manual	90 15 25
Xerox FORTRAN/Library Technical Manual	90 10 36
Xerox ANS FORTRAN IV/LN Reference Manual	90 18 06
Xerox ANS FORTRAN IV/OPS Reference Manual	90 18 07
Xerox Sort/Reference Manual	90 17 87
Xerox Report Program Generator (RPG)/Reference Manual	90 18 41

Manual Content Codes: BP - batch processing, LN - language, OPS - operations, RP - remote processing, RT - real-time, SM - system management, TS - time-sharing, UT - utilities.

CONTENTS

APPENDIXES

FIGURES

TABLES

GLOSSARY

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 !XEQ control command.

area: a contiguous portion of a random access device that contains files of some related nature.

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 when no interrupts are active. These programs are entered through the batch processing input stream.

batch processing: a computing technique in which similar programs are grouped together and processed or executed in a single run so as to effect efficient utilization of the computer.

channel status table: a table of eight words per SYSGEN-defined I/O channel that reflects the hardware condition of each I/O channel.

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 at its interrupted point.

clock counter: a memory location that records the progress of real time or its approximation, by accumulating counts produced by a (clock) count pulse interrupt.

close: terminating the use of an item (such as a file) and performing certain clean up operations to provide for its future reuse or the reuse of its resources.

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.

count zero interrupt: an interrupt level that is triggered when an associated (clock) count pulse interrupt has produced a zero result in a clock counter.

critical task: a task whose importance is high enough that no attempt should be made to run without it in the event of a serious error.

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

device-file number (DFN): a logical method of referring both to a physical peripheral device and to a collection of information about the device. The device file number indicates the order in which devices are initially defined at SYSGEN. For example, the first device defined must always be a keyboard printer (DFN 1).

device name: an identifier used at SYSGEN time for an actual physical I/O device that is composed of two elements: a device type which is a two-character code for a particular class of peripheral devices, and a device number which is a two-digit hexadecimal representation of the physical unit number associated with a device.

device unit number: an integer value coded into a FORTRAN IV program to reference peripheral devices. Standard device unit numbers can be equated to device file numbers (see above) either at SYSGEN time or through !ASSIGN commands.

directory: a table of names and addresses of files on a random access device that enables the system to locate a file when given only its name and area.

disabled: the condition of an interrupt level wherein the level may advance from the armed to the waiting state when triggered by an interrupt pulse, but the level cannot cause a program interruption until it is enabled; it thus remains in the waiting state until it is allowed to interrupt the program.

disarmed state: the state of an interrupt level that cannot accept an interrupt input signal.

disk pack: a secondary storage system of removable rotating memory. For most RBM purposes, disk pack and RAD are synonymous unless otherwise noted.

enabled: the condition of an interrupt level wherein the level is not inhibited from advancing from the waiting state to the active state except for priority considerations.

end action: that action that takes place at the completion of an I/O operation. This usually includes the entry of a special routine that was specified when the request was made.

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

error severity level code: a code indicating the severity of error 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 that changes the address at which program loading is to begin.

external interrupt: one of the class of interrupts that are associated with special systems equipment. These interrupts are "external" to the basic computer system and are associated with functions that are defined according to the requirements of a particular installation.

external interrupt inhibit: the bit, in the program status doubleword, that indicates whether (if 1) or not (if 0) all external interrupts are inhibited.

external reference: a reference to a declared symbolic name that is not defined within the 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 module.

file control table: contains information about all device files in the RBM system and is indexed by device-file number.

file name: a name for a permanent file that is defined either at SYSGEN or later through the RAD Editor.

flawed track: a disk pack track that contains a flaw mark in the header as well as the address of an alternate track.

foreground area: that portion of memory dedicated specificially for RBM, service routines, and foreground programs.

foreground program: a program that executes in the foreground area of core and can utilize all privileged services.

foreground task: a body of procedural code that is associated with (connected to) a particular interrupt.

GO file: a RAD file of Relocatable Object Modules (ROMs) formed by a processor. This is a default input file when no file name is specified.

granule: a record beginning on a physical sector boundary, used as a unit of allocation for random RAD or disk pack files. A granule is usually synonymous with a sector on a device, but may be defined (on a file basis) to be equivalent to a partial sector, one sector, or several sectors.

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

inhibited interrupt: a condition of an interrupt that prohibits it from entering the active state.

input/output interrupt: an interrupt triggered by the standard I/O system of the computer.

installation input parameter: any input parameter used during System Generation to direct the formation of an RBM system.

internal interrupt: one of the class of interrupts that are supplied with a standard computer system, or are optional additions associated with dedicated functions (such as power fail-safe). These interrupts are "internal" to the basic computer system.

interrupt trigger signal: a signal that is generated, either internal or external to the CPU, to interrupt the norman sequence of events in the central processor.

I/O block: a contiguous amount of RAD or disk space that contains records of blocked or compressed files. All I/O blocks are the same size (K:BLOCK) and always begin on a sector boundary. K:BLOCK also specifies the size of core blocking buffers.

I/O control table: a table containing the device-specified input/output control doublewords and other information necessary for RBM I/O services. There is a one-to-one correspondence between the I/O control table and file control table.

I/O control subtable: same as I/O control table except that the subtable is RAD specific.

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

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

link editing: the process of combining separately compiled or assembled program modules, relocating them, linking them to defined library routines, and producing an absolute executable load module.

loading: the process of reading an executable program (see link editing above) from secondary memory to absolute locations in main memory.

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. Or, a SYSGEN mechanism for reserving logical groups of DFN's for a combination of foreground and background use to accomplish information transmission between tasks without the use of any real peripheral device.

1. INTRODUCTION

RBM CHARACTERISTICS

The Xerox 530 and Sigma 2/3 Real-Time Batch Monitor (RBM) is the major control element in the operating system. It supervises and services simultaneous foreground programs and background batch programs without interfering with the real-time response capability of the foreground.

RESIDENT SECTION

The resident portion of RBM consists of the following parts:

- Several independent tasks that are connected to the hardware interrupts (e.g., the real-time tasks). The tasks are not reentrant. They can communicate with each other and may use some of the monitor service routines.
- Several reentrant monitor service routines that can be used by any task in the system. These are described in Chapter 4.
- Standard system constants and tables (see Appendix C).
- Input/output tables for constants and status information.

NONRESIDENT SECTION

The nonresident part of RBM consists of the system initialization portion that is loaded at the time the system is created, monitor service routines, and device-dependent I/O routines for which a response is not critical. The initialization portion selects the optional features of RBM and initializes the input/output constants.

SYSTEM ENVIRONMENT

In addition to the Monitor itself, the hardware-software environment of the operating system consists of the following major elements:

- Xerox Model 530 or Sigma 2/3 computer system including (a) the required system RAD, (b) the selected number of hardware interrupts connected to various foreground tasks in user-determined priority sequence, (c) dedicated and commonly shared I/O devices.
- Partitioned core memory (see Figure 1) divided into
  - A protected RBM area reserved for the RBM monitor.

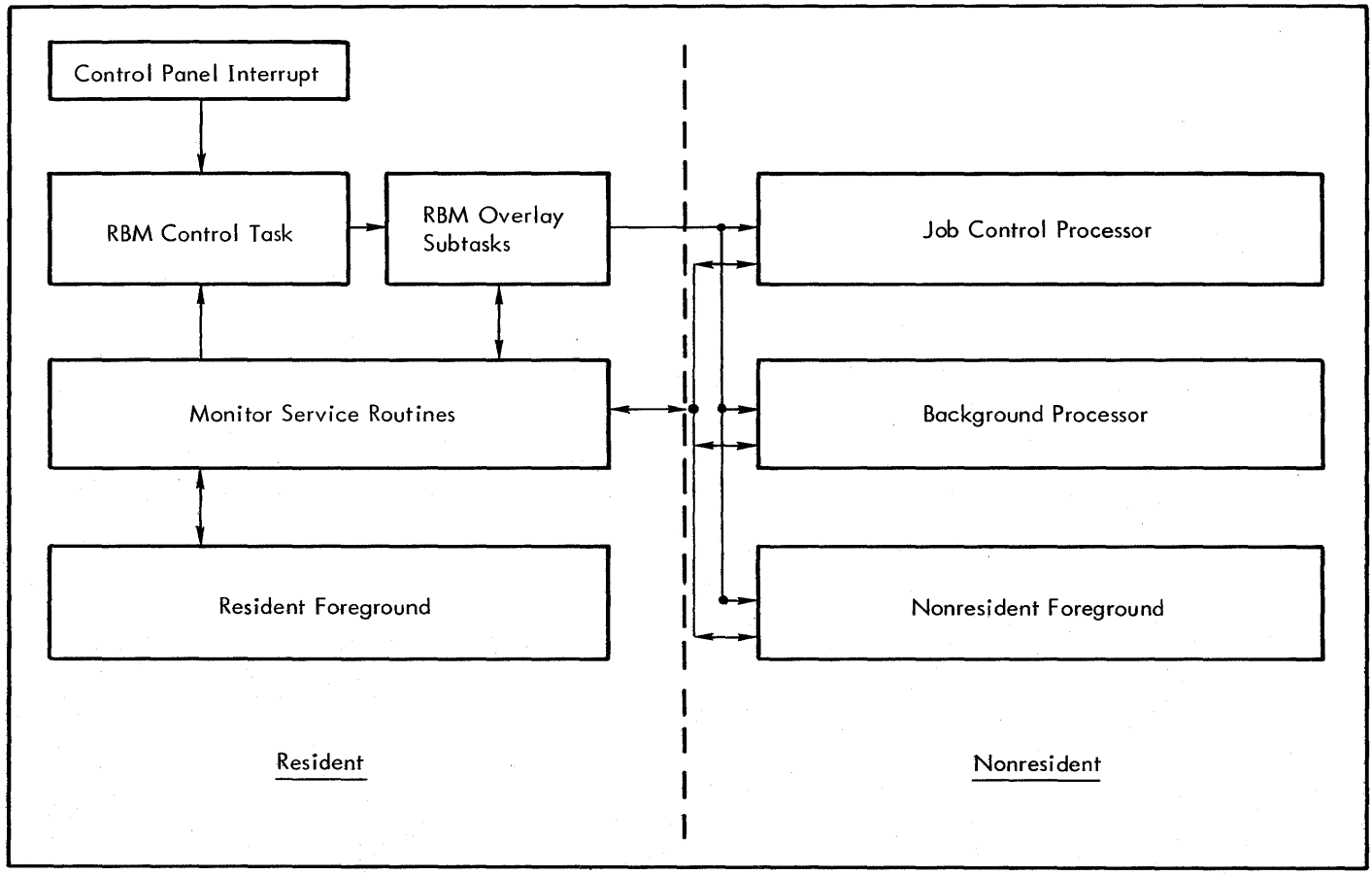


Figure 1. Operating System

logical record: a record that is a fixed measure of contiguous data (on a file basis), distinctive as being meaningful to the user. For blocked RAD files, logical records are contiguous within blocks but need not be integral to a block.

memory protection: the use of the optional protection feature that keeps unprotected background memory from altering protected foreground meaning.

memory write lock: a one-bit write-protect field optionally provided for each 256-word page of core memory addresses.

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

nonresident foreground program: a foreground program explicitly called from secondary memory that resides in the nonresident foreground area of core memory during execution. The space thus occupied is considered "active" and is protected by the Monitor from interference by other activities.

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.

open: the preparing of an item (such as a file) for initial use.

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

operational label table: there are two tables: one for foreground and one for background. The tables contain the two-character operational labels that are used for reference by the RBM service routines and connect an operational label to a device file number.

option: an elective operand in a control command or procedure call.

Overlay Loader: a processor that links and absolutizes elements of 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.

OV file: a 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. This is a default file when no output file is specified.

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

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

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

priority level: priority level of a task is dependent on the position of its associated hardware interrupt in the priority chain.

RAD/disk areas: the allocation and definition of a RAD into specific areas during SYSGEN, each of which is labeled with a two-character mnemonic to expedite file management.

Rapid Access Data (RAD) storage system: a secondary storage system of rotating memory. For most RBM purposes, RAD and disk pack are synonymous unless otherwise noted.

real-time processing: data processing designed so that the results of the operations are made available in time to influence some process being monitored or controlled by the computer system.

reentrant: that property of a program or subroutine that enables it to be interrupted at any point, employed by another user, and then resumed from the point of interruption. Reentrant programs are often found where there is a requirement for a common store of public routines that can be called by any user at any time. The process is controlled by the Monitor which preserves the routine's environment (registers, working storage, control indicators, etc.) when it is interrupted and restores that environment when the routine is resumed for its initial user. A reentrant routine never stores any intermediate values within itself.

Relocatable Object Module: a program or subprogram that may be relocated and link edited to operate anywhere in core; that is, does not have absolute addressing.

resident foreground program: a foreground program that is automatically loaded into a fixed area of foreground core memory every time the system is booted in.

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 or disk pack).

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

semiresident foreground program: a foreground program explicitly called from secondary memory that resides in the resident portion of core memory during execution.

service routines: Monitor-supplied services and operations that can be called by an executing foreground program, or else by an executing background program (except for certain privileged function dedicated to foreground use).

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.

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 relocatable object language format, any of which may be included in a program being created.

Task Control Block (TCB): part of the load module that contains the area required for context storage. The TCB is task-associated.

temporary files: those files that exist only until the current job step ends. They may, or may not, have existed prior to the start of the job.

Temp Stack: an area of memory optionally created by the Overlay Loader for a user program and used by the Monitor and System Library routines.

unsolicited key-in: information entered by the operator via a keyboard in response to a Control Panel interrupt.

- A protected resident foreground area reserved for user foreground tasks.
- A protected nonresident foreground area reserved for a single nonresident foreground program.
- A protected public library area reserved for public library routines shared by foreground and background tasks.
- An unprotected background area used by background (non-real-time) processors, translators, and batch users' programs, and occasionally by foreground programs requiring temporary use of additional memory. (In this case the foreground will checkpoint the background.)

- The system RAD,[†] allocatable into permanent and temporary files. The permanent files contain all the background RBM processors such as Basic FORTRAN IV or ANS FORTRAN IV, Extended Symbol, RAD Editor, etc., plus RBM itself. They may also contain user data and operational resident and nonresident foreground programs that can be called into protected memory for processing. Temporary files are normally used as intermediate scratch areas by processors or user programs.
- A number of user foreground tasks that can be connected to hardware interrupts. Examples of foreground tasks are process control operations, real-time data acquisition and control, and low-speed telemetry applications. The RBM Control Task is connected to the lowest priority hardware interrupt in the system so that no background processing can delay foreground tasks.
- Overlay Loader for linking and absolutizing segmented foreground and background programs that enables background processors and user programs to overlay themselves in core storage, and thus permitting programs of virtually unlimited size to be executed.

FOREGROUND (High-Level Priority Response)

Within the framework of the user-determined hardware interrupt priorities, foreground programs or tasks operate as independent entities, and the Monitor generally makes no attempt to interject itself between these tasks and their real-time functions. The Monitor services the foreground only on request, such as a call to one of the monitor service routines. The principal foreground services of the Monitor are to

- Respond to I/O interrupts.
- Respond to an operator's console request (such as queuing).
- Supervise RAD file activity.
- Optionally, supply a software version of multiply/divide functions for configurations without multiply/divide hardware.
- Load a foreground program into memory from the RAD on request.
- Provide the foreground with standard constants (see Appendix C).
- Make available a "mailbox" area of 32 memory locations for communication between two or more foreground programs.

The interrupt priority sequence (described in detail in the respective computer reference manual) is the basis for the priority level of tasks in the RBM system. That is, the priority level of a task is dependent on the position of the associated hardware interrupt in the interrupt priority chain. Background jobs in the system all have the same priority level. A background job is not connected to any interrupt level in the system, i.e., its priority is below all hardware interrupt levels and is processed serially.

BACKGROUND (Low-Level, No Priority)

The primary function of the Monitor is to supervise and control all those operations that take place in the unprotected background area by the following means:

1. Use only available foreground idle time for background processing.
2. Interpret control functions from control command card images via the Job Control Processor.
3. Supervise the loading and execution of all background jobs and activities in unprotected memory.
4. Provide simple background scheduling (first-in, first-out).
5. Provide I/O services for the background job stack.
6. Inform the operator on the status of peripheral device operations.
7. Test all background operations and processes for foreground protection violations and prevent the background from altering or delaying foreground response or from using dedicated I/O devices.

RBM processors and permanent user processors may be loaded onto permanent RAD files and then executed by control command. Programs may also be loaded onto temporary RAD files for the duration of the present job.

All programs must exist on the RAD in absolute core image form for execution. Relocatable programs, consisting of a root and one or more overlay segments linked by external references, must be created by an Overlay Loader to link all modules and create the proper overlay structure for execution.

[†]For RBM purposes, RAD and disk pack are synonymous unless specifically stated otherwise.

It is possible to create programs consisting of a root and one or more overlay segments through use of the Absolute Loader if there are no external references (see the !ABS command in Chapter 2 for other restrictions).

Two levels of logical (rather than physical) device referencing are provided, enabling system configurations to change or expand without reprogramming. Further, through many device-independent features and use of standard media formats, input and output can be directed to card equipment, paper tape equipment, or magnetic tape without changes in the user's program.

For maximum flexibility and control of input/output, the user can optionally specify his own I/O Control Doublewords and order bytes, perform independent error recovery, and be informed by RBM when an I/O operation has terminated. Alternatively, for greater ease of programming and device independence, the RBM will create the IOCDs and order bytes and perform standard error checking and recovery.

When multiprogramming with foreground tasks and background jobs, the foreground has access to all privileged instructions. The background is checked by both hardware and software to provide complete protection of a foreground program's use of core memory and peripheral operations.

SECONDARY STORAGE MANAGEMENT

The RBM operating system provides use of the RAD or disk packs for

- Temporary and permanent files.
- User and system files.
- Sequential files (pseudo tape, where RBM performs all file management).
- Random-access files (RBM performs I/O transfer and controls file limits, but user controls relative addressing).

RAD/DISK PACK AREAS

The concept of RAD/disk pack areas is a convention created primarily to expedite file management. RAD and disk pack areas are allocated during system initialization. Disk pack areas may also be allocated after system initialization, using the RAD Editor. The areas are labeled with two alphanumeric characters, from the following list:

SP	BT	BP	Xn
SD	CP	UP	αα
SL	FP	UL	

where n is a decimal digit and α is any letter combination except Xn. Note that the combination "SK" has a special meaning (described in "SYSGEN" chapter of Reference Manual 90 30 36) and is not an area mnemonic.

The labels have the special meaning given in Table 1.

Table 1. RAD/Disk Areas

Mnemonic	Meaning
SP†	System Processor area. Contains RBM and user-selected processors from the list given in Table 5 (the Overlay Loader is a mandatory processor). This area is searched whenever either a system processor or user processor is requested.
SD†	System Data area. Contains files necessary for the execution of RBM.
SL†, UL	System Library and User Library areas. These are the only areas from which the Overlay Loaders will load library routines.
UP, FP, BP	User, foreground, background processor areas. Contains resident foreground programs, foreground tasks, nonresident programs, semi-resident programs, and background programs. Area FP may only contain files of foreground or no-write protect codes and area BP may only contain files of background or no-write protect codes. The UP area may have its area write protection specified during System Generation or RAD Editing. These areas and the SP area are searched when a processor is requested. SP, UP, and FP are searched (in the order given) for resident foreground programs, when the system is booted from the RAD.
BT†	Background Temp area. Used for allocation of temporary files.
CP†	Checkpoint area. Used to store the background environment when a background program is checkpointed by a foreground process.
αα	User data areas which contain any data the user desires, including program files.
Xn	Xn areas are similar to αα areas except that the user has the option to perform his own management of the entire area, thus allowing access to data arranged in nonstandard formats. No disk pack verification is performed for an M (Mount) key-in (see "M Key-Ins" in Chapter 3).

†These areas receive default allocations during SYSGEN. Note that the SP and SD areas must be present in the system

PROCESSOR FILES

Processor files are stored either as a single segment or as an overlay structure. Overlay Loaders store the files on the RAD in core image form, ready for loading, and absolutized for the space they will occupy at execution. The processor files are loaded for execution via a processor control command.

LIBRARY FILES

Library files contain subprograms in a relocatable form. The files have specified entry points and are in the form of binary card images in Standard Object Language.

There is one library file for the system area mnemonic SL, and one for the user area mnemonic UL. Overlay Loaders can load selectively from one or both, in either order or priority. Although records within a subprogram are loaded sequentially, access to the individual subprogram is on a random (direct access) basis.

DATA FILES

Permanent data files may contain any kind of data and may be accessed sequentially or randomly, depending on how they were created. The user is responsible for reading them accordingly.

FILE NAME

Only permanent RAD files have a file name. Some names are entered into the dictionary for the appropriate area at System Generation; others are entered later by the RAD Editor. After the name is in the dictionary, an !ASSIGN control command or a call to M:ASSIGN can equate either an operational label or a FORTRAN device unit number to this file name.

OVERLAY CAPABILITIES

Under RBM, Overlay Loaders can be used to create overlay programs for later execution in either the foreground or background.† The overlay programs can be permanently entered (as a file) into either the System or User Processor areas, or into a temporary overlay file (OV). Since they are stored on the RAD in absolute core image format, they can be quickly loaded into memory for execution.

Each segment is created by an Overlay Loader from one or more object modules (assembly language, FORTRAN, or library routines). The control commands required to create the overlay segments are defined in the discussion of the Overlay Loaders. During execution, the Monitor service routine M:SEGLD is used to control both the loading and the transfer of control between various segments.

†For a complete description of the Overlay Loaders, see the Overlay Loaders chapter.

TASK DISMISSAL

The dismissal option allows foreground tasks to be automatically dismissed by RBM when they would otherwise be waiting at a high level for on-going I/O to complete. This feature allows automatic overlap of high level (i.e., foreground) I/O and low level CPU execution to the enhancement of low level throughput. The feature is controllable on a task basis or a system basis and requires very little core space.

CHECKPOINT/RESTART

The checkpointing feature permits a partially processed background job to be saved in secondary storage along with all registers and other environment. The vacated background space is set to protected status and is then available to the interrupting foreground task for either instructions or temporary data storage.

Checkpointing ensures continuity to the partially completed background job by not repositioning any background peripheral devices, permitting all current background I/O activity to complete, and writing all of the background space onto a prespecified RAD area.

Restart takes place when the previously checkpointed background program is reloaded from the RAD and continues execution as though the interruption never took place.

PUBLIC LIBRARY

Most of the support on FORTRAN and mathematic routines are reentrant.† If an RBMsystem has several real-time foreground tasks that use a number of the same subroutines, the collectively-used set of subroutines can be loaded together into what is termed a Public Library. Thereafter, whenever the Overlay Loader processes a foreground or background program that references one of the "public" routines, it sets the appropriate branch to the Public Library. The Public Library is loaded into core whenever RBM is rebooted from the RAD.

When one of the Public Library routines needs temporary scratch space, it requests space (via a call to M:RES) from the temporary stack of the task that is calling the Public Library routine. When the library routine exits, the space is released via a call to M:POP.

†See the ANS FORTRAN IV Library Technical Manual, Publication No. 90 18 35 for restrictions concerning library routines in the Public Library.

REENTRANT ROUTINES

In RBM usage, "reentrant" means that a subprogram (never a task) may be interrupted during execution, called again by the interrupting task, and later reentered and continued from the location of the former task. This is a last-in, first-out kind of reentrancy in keeping with the computer's priority interrupt system.

ACCOUNTING AND ELAPSED TIME

Background job accounting and provisions to limit the execution time of a background job can be accomplished by specifying the JOBACCT option at SYSGEN. To correctly calculate the elapsed time for the background, the Monitor M:SAVE routine changes the charge index to foreground at the first interrupting foreground task. M:EXIT restores the charge index when return to background is sensed.

JOBACCT is also used to limit the execution time of a background program. The user may limit this execution time by using the !LIMIT control command, and Clock 1 will provide watchdog services on the background program.

When a !JOB control command is read, an entry is created in the accounting file (RBMAL,SD). The entry includes the start time, user name, and account number. The start time of the job is then logged on the LL device as mm/dd/yr hrmn.

At the completion of each activity, the accumulated elapsed time of background execution will be logged on the LL device as

ET=mmm.mm (minutes)

At the completion of the job (i.e., a new !JOB or !FIN command) the current date and time and a job recap are logged on the LL device as

mm/dd/yr hrmn BK=mmm.mm,

FG=mmm.mm, ID=mmm.mm

where

BK represents the total job time. The total time for a job is defined as the time available to the background from the time the !JOB control command is read until the next !JOB or !FIN command is encountered.

FG represents the amount of time used by interrupting foreground tasks during the job.

ID represents the accumulated idle time incurred within the job. This could be a result of an M:WAIT request, a W key-in, !PAUSE command, or an attended job being aborted.

The time for a background job is recorded in the accounting file entry for that job. The IDLE account is updated to reflect total idle time charges. After the !FIN control command is read, all idle time is charged to the IDLE account.

The following rules govern the operations of the Accounting Log:

- A call to M:SAVE switches from the background to foreground time accumulation.
- A call to M:EXIT switches from foreground accumulation to background accumulation if a background job is executing.
- A W key-in, M:WAIT request, or !PAUSE command switches from foreground accumulation to idle time accumulation. An abort from an attended job switches the same way. An S key-in switches back to foreground accumulation from the idle accumulation. The M:EXIT to background switch charges to background.
- A !JOB or !FIN command writes out total accumulated times and resets times to zero.
- The ET (elapsed time) is printed on LL each time JCP is read into the background and represents the total elapsed background execution time.

SYSTEM INITIALIZATION AND CREATION

The RBM system is created for a particular installation through a nonpermanent system generation (SYSGEN) program (see System Management Reference Manual 90 30 36).

The user defines RAD areas, optional routines, peripheral devices, and operational labels. This is followed by a definition of the exact bounds on the foreground, monitor, and background memory areas, and the size of the RAD areas.

Once the system is completely defined, the required routines are loaded and a rebootable version is written onto the RAD.

If the system must be restarted later, the rebootable version is loaded from the RAD. A completely new system initialization is necessary only if the above mentioned definitions must be changed.

When the system is created, a version number is specified that will be printed on LL at the beginning of each job for reference.

Most of the Xerox disk devices have manual switches that may be used to permanently protect certain areas.

RBM SUBSYSTEMS AND PROCESSORS

RBM supports the subsystems, processors, and foreground facilities described below. The subsystems and processors execute in the background area of core memory.

STANDARD SUBSYSTEMS

OVERLAY LOADERS

The Overlay Loaders form absolute binary overlay segments for later execution in either foreground or background areas. If a resident or nonresident program can tolerate a loading delay of 20 to 100 ms, foreground or background programs of virtually unlimited size can be constructed with the Overlay Loader despite limitations in available core storage.

RAD EDITOR

The RAD Editor performs RAD allocation for permanent files and generates and maintains directories for the permanent RAD areas: System Processor area, System Library area, System Data area, User Processor area, User Library area, User Data area, and any $\alpha\alpha$ areas and Xn areas. It allows dumping of files and mapping of all RAD areas, including checkpoint and temporary areas.

UTILITY SUBSYSTEM

The RBM Utility subsystem provides a universal media copy routine, object module editor, dump routine, and record editing by line or sequence number.

LANGUAGE AND SERVICE PROCESSORS

EXTENDED SYMBOL

The Extended Symbol assembly language processor (assembler) provides upward compatibility with basic Symbol plus extended capabilities that include using the RAD for overlay to reduce core residence requirements.

The processor accepts as input a source program coded in either Symbol or Extended Symbol, processes it, and outputs an object module, diagnostic messages, an optional assembly listing, and an optional cross-reference listing.

BASIC FORTRAN IV

Basic FORTRAN IV is a one-pass compiler with capabilities extended beyond Basic FORTRAN. It can compile large source programs by using the RAD for overlay to minimize core residence requirements, and has two floating-point modes: standard precision and extended precision.

ANS FORTRAN

The Xerox ANS FORTRAN IV compiler provides a full FORTRAN IV capability. ANS FORTRAN IV is designed for real-time reentrant usage, as well as for normal batch processing. It is upwards compatible with the Basic FORTRAN IV language compiler. It meets and exceeds the specifications given in the ANSI FORTRAN X3.9-1966. This expanded version of the compiler adds language syntax sophistication that both simplifies problem solving and allows for greater programming flexibility than earlier versions of FORTRAN IV compilers. (Not available with Sigma 2 systems.)

RPG

RPG allows users to perform batch data processing tasks using simplified programming techniques. Xerox RPG is an expanded version of conventional RPGs that accepts and processes IBM 1130, 1800, and 360/20 RPG specifications. RPG is useful for any installation with a need to

- Create reports. The fixed program logic of RPG is ideally suited for those installations that require one-time reports, since elaborate coding is not required for generating those reports.
- Process inventory, payroll, or other commerical applications.

Under Sigma 3, RPG requires the Extended Arithmetic (8119) feature. (Not available with Sigma 2 systems.)

SORT

The Xerox Sort processor offers a generalized file sorting capability. Xerox Sort is a disk-oriented multiphase program that is overlayed to reduce memory requirements. The sorting technique used is a replacement-selection tournament with a balanced merge of intermediate strings. (Not available with Sigma 2 systems.)

COBOL

Xerox 530 ANS COBOL offers a powerful and convenient programming language for implementation of business or commercial applications. Xerox 530 ANS COBOL is a subset of the X3.23 - 1973 ANS COBOL Standard and contains the following modules implemented at the first level:

- Nucleus
- Table Handling

- Sequential I/O
- Relative I/O
- Indexed I/O
- Inter-Program Communication
- Library
- Debug

Additional features such as in-line diagnostics optional data map, procedure map, object listing and cross reference are included.

The Xerox 530 ANS COBOL compiler is a two-pass processor using a segmented structure to minimize the core required for operation. The compiler runs in the background under control of RBM.

Sequential, Relative and Indexed files produced by 530 COBOL are compatible with those produced by 530 RPG II Version C00.

Under Sigma 3, COBOL requires the Extended Arithmetic (8119) feature (not available with Sigma 2 systems).

OPTIONAL FOREGROUND FACILITIES

DEBUG

The RBM Debug package provides the user with a debugging tool designed primarily for nonsegemented background programs but with a limited capability for debugging foreground programs. The Debug functions and commands are described in Chapter 12.

COC HANDLER

The character-oriented communications (COC) handler provides communication between real-time programs and various terminal devices. The COC hardware consists of a controller and from one to eight transmission-line interface units. RBM can accommodate one COC controller. See Chapter 4, M:COC, for a more complete discussion of the COC handler.

PLOTTER SYMBIONT

The plotter symbiont is a foreground program that drives a Xerox 7530 or 7531 graph plotter via a symbiont file on disk. The FORTRAN subroutine library provides background program subroutine calls for building the plotter-symbiont command file.

XSP

The Xerox Satellite Processor (SP) provides Xerox 530 or Sigma 3 computer sites with a capability for high-speed telecommunications with other host remote computer systems. Operating under either a Xerox 530 or Sigma 3 operating system, the Xerox Satellite Processor permits communication with any host Xerox computer running under the Control Program-Five (CP-V) operating system and non-Xerox host computers in accordance with the HASP Multileaving Protocal.

The basic function of the Satellite Processor is to move streams of sequential data from source devices or files to destination devices or files at the request of the operator, which provides the Xerox 530 or Sigma 3 user a highly convenient means for utilizing the full resources of a larger host system or exchanging data with another HASP compatible workstation. Remote activities may take place concurrently with local foreground and background processing, subject to device and resource availability. Spooling of remote data using magnetic tape is supported. (Alternately, a sequential disk file may be substituted for magnetic tape.)

A Satellite Processor site can communicate with three general classes of remote sites:

1. CP-V host.
2. IBM host.
3. Another Xerox workstation or IBM HASP compatible workstation.

Support for up to four 7605 communications controllers is provided. A single HASP host may regard the Xerox Satellite Processor as one to four workstations. Or, one to four HASP hosts or HASP compatible workstations may each regard the Xerox Satellite Processor as a single workstation. Any combination thereof concurrent with a single spooled playback operation is also allowed.

The Xerox Satellite Processor will address a data set controller exclusively in half-duplex mode in order to realize the increased line-driving efficiency provided by software command chaining. The 7605 controller is used by Xerox Satellite Processor in either two-wire or four-wire mode. Line speeds supported are 2000 - 19,200 bits per second.

BSS

A low overhead Basic Spooling System is provided for RBM. The Basic Spooling System is intended to provide minimum core resident support for RBM users. Basic Spooling System functions are as follows:

- Spools to circular direct access disk file.
- Unspools to a physical or logical device.
- Operator may start, stop, suspend, skip, or backspace.

- Overflow threshold alerts operator when critical low available space conditions occur.
- Core resident control information is periodically checkpointed to disk resident information table to prevent loss of data.
- If any record with the characters *FORMS in column 1 through 6 is detected, the *FORMS record is diverted to the Operator's Console and an automatic STOP occurs.

RBM TERMS AND PROCESSES

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

TASK

A "task" is an entire set of foreground operations performed independently of other tasks in the system. It must be connected to one and only one hardware interrupt. A task may use Monitor service routines but must never branch to another task. One task may trigger the interrupt level of another task by means of a Write Direct instruction. The prescribed entrance and exit procedure for all real-time tasks in the system is described in Chapter 6.

A task logically consists of three parts (that may or may not be contiguous in core storage):

1. A Task Control Block (TCB) that contains status information and the contents of the registers from the interrupted task (see Table 18). The TCB is normally the first loadable item in the object module.
2. A task body, consisting of a sequence of instructions executed in response to the task interrupt.
3. A task temporary storage area for use by the Monitor service routines (and other reentrant library routines) to provide reentrancy for these routines.

Examples of foreground tasks are

- Real-time foreground tasks connected to external interrupts.
- Monitor I/O interrupt routine.
- Monitor Control Panel interrupt routine.
- Monitor machine fault and protection violation routines.
- RBM control routine (for loading, abort, etc.).

A background program can also operate as a single task but without foreground privileges.

PROGRAM

A "program" is one or more tasks (and optionally, some data storage) that are loaded and controlled as a unit. Four types of programs exist under RBM:

1. Resident foreground programs consisting of one or more tasks, perhaps some special routines for receiving I/O interrupt responses (see "End Action"), and any common storage that may be needed.
2. Semiresident foreground programs that are explicitly called in from secondary memory and reside in the resident portion of core memory during execution.
3. Nonresident foreground programs.
4. Background programs, consisting of a single task.

FOREGROUND

"Foreground" refers to real-time or Monitor tasks executed in protected memory on a real-time basis. Since the number of foreground tasks is limited by the number of internal and external interrupts available in the system, the fundamental limitation is the amount of core space available. However, the use of overlays and nonresident foreground programs makes the amount of effective foreground space virtually unlimited, depending only on the severity level of required response times.

BACKGROUND

"Background" refers to a non-real-time program executed in available nonprotected memory. The purpose of background programming is to achieve higher efficiency in the system by using the CPU time not needed by real-time tasks to maintain foreground programs, or to perform other data processing functions.

Background operations may be assemblies, compilations, data processing, or utility operations. The two fundamental restrictions in using background programming are

1. A background program is never allowed to interfere with real-time foreground tasks, it must operate in nonprotected memory and use the Monitor service routines for all I/O and other privileged operations.
2. Since a background program uses only the CPU time available after the real-time foreground is satisfied, it

may not be guaranteed any CPU time when foreground is very active. The background cannot inhibit interrupts or do anything else that might interfere with real-time foreground responsiveness.

JOB

A "job" is defined as consisting of all background activities or processes that take place between a !JOB command and the next !JOB command or a !FIN command (whichever is encountered first).

JOB STEP

A "job step" is defined as the operations performed in setting up and processing a single program within a job stack. A job step is initiated by calling in a background processor and ends when the processor exits.

MONITOR SERVICE ROUTINES

RBM service routines can be used by real-time foreground tasks, a background task, or RBM tasks. All routines are coded in a reentrant manner, and those that require temporary storage use the temporary stack space associated with the task that calls the routine (see Chapter 4).

TEMPORARY STACK

The temporary stack (temp stack) is a block of core storage associated with a particular task and is used by Monitor service routines for temporary storage to achieve reentrancy. An entry in the TCB for a task points to the temp stack space. When a task is active and using either Monitor service routines or the floating accumulator (defined below), the beginning of the temp stack space for the active task must be set into core memory location 6 (after the previous contents of location 6 are saved). Monitor service routine M:SAVE will set this pointer.

When Monitor service routines or Public Library routines need temporary space, they can call M:RES to reserve space, and M:POP must then be called to release the space when it is no longer needed. Thus, the total temp stack is a function of the deepest nesting of calls to Public Library routines and RBM service routines and of the space required for these routines.

FLOATING ACCUMULATOR

This software convention is used extensively by mathematics library routines and can also be used by any user's program. The floating-point accumulator is assumed to occupy the first six locations of the temporary stack space. It is used like a hardware accumulator, i.e., to build up a cumulative result from single-precision or double-precision real (floating-point) calculations.

As a convenience in referencing the floating accumulator, core locations 1 through 5 are set with pointers to the actual core locations. This is done when entry is made to the active task (by M:SAVE when the routine is used). Therefore, indirect addressing through locations 1 through 5 will result in storing, loading, or modifying the actual floating accumulator. The sixth cell of the floating accumulator is used by the FORTRAN-formatted I/O routine.

RBM CONTROL TASK

The RBM Control Task encompasses a number of subtasks that control the reading of control commands, loading background programs, interpreting unsolicited key-ins, and aborting or terminating a background job. During system initialization, the RBM Control Task must be assigned to the lowest priority hardware interrupt.

The RBM Control Task uses the same entrance and exit procedure and the same type of TCB as a real-time foreground task. Since its main function is to control background activity, it has a lower priority than any real-time task. It is necessary that this be a separate task (and not part of the background priority level) so that effective and responsive control can be made through key-ins. All RBM functions associated with this level operate as subtasks to the RBM Control Task and are non-reentrant.

NONRESIDENT FOREGROUND

Nonresident foreground programs are real-time programs not needed in core on a continuous basis. They are created like resident foreground programs and are then written on the RAD in the user processor (UP) area. An operator or a resident real-time program can later call one of these nonresident programs, and it will be loaded and executed like a permanently resident real-time foreground program with all the protection and priority privilege characteristics of the foreground.

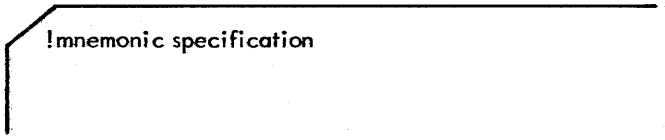
COMPRESSED RAD FILES

EBCDIC character codes do not use all possible bit combinations of an eight-bit byte, and some combinations (X'DC' and X'EC') are therefore available for special coding bytes. Since EBCDIC information often contains a large number of "blank" byte strings, a code and a word count are used to replace an entire string of blanks. Thus, several 80-byte source cards (usually about 12) can be compressed and blocked into a 360-byte RAD sector. The RBM Read and Write routines provide the compression or decompression feature, and the user program can read or write as though the file contained 80-byte card images. Compressed files are always blocked; that is, several records are transferred with one RAD access.

2. CONTROL COMMANDS

The Monitor is controlled and directed by control commands that initiate loading and execution of programs and provide communication between a program and its environment. The environment includes the Monitor, background processors, the operator, and peripheral equipment.

Control commands have the general form:



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. It must immediately follow the ! character without intervening spaces.

specification is a listing of required or optional specifications. This may include labels and numeric values appropriate to the specific command. In the specification field, hexadecimal values must be shown as +xxxx and EBCDIC values must begin with a letter; any other values are assumed to be decimal values. Specification fields are separated by a comma or an equals sign.

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 although brackets are not used in actual control command format.

One or more blanks separate the mnemonic and specification fields, but no blanks may be embedded within a field. A control command is terminated by the first blank after the specification field. Annotational comments detailing the specific purpose of a command record may be written following the specification terminator, but not beyond column 72. Only columns 1-4 are examined to determine control functions; only the first eight nonblank characters following the ! are used to locate processors.

The user may insert comment lines within a job stack at any point where a Monitor control command would be recognized. A comment line contains an asterisk as the first character of the line. The comment line is listed on the LL device.

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 (see !ASSIGN command). Control key-ins are always input through the keyboard/printer. All control commands and Monitor messages are listed on the output device designated as the listing log (normally a line printer) to provide a hard-copy history of a job.

JOB CONTROL PROCESSOR (JCP)

Monitor control commands are read from the background operational label CC unless the operator has requested a keyboard/printer override through an unsolicited KP key-in. All such commands are read by the Job Control Processor (JCP), a special processor loaded into the background by the RBM and reloaded into the background following each job step within a job. When a control command is encountered by the JCP, the order of search is

1. Monitor control commands.
2. System processor names.
3. User processor names.
4. Foreground processor names.
5. Background processor names.

A !JOB command sets all background operational labels to their standard assignments. All temporary RAD space is set "unused" and is then available for following job steps.

As the JCP encounters !ASSIGN and !DEFINE commands between job steps, it makes appropriate entries in the operational label tables and continues to do so until it encounters a request for a processor. When the requested processor is read into the background and attains control, this marks the beginning of a job step.

At the end of each job step (i.e., when the JCP begins reading control commands at the completion of the previous job step), all background operational labels associated with temporary RAD space are set to an undefined status and all temporary background space is reset to an "unused" status unless a !TEMP S control command is in effect, which saves temporary files until a !TEMP R, !JOB, or !FIN command is encountered.

MONITOR CONTROL COMMANDS

ABS The !ABS control command causes the Absolute Loader to read absolute binary programs from the AI device and write core image copies onto the OV file. The last (or only) segment to be read must be followed by an !EOD command. The binary program(s) following the !ABS command must contain only those load items that are part of the standard absolute object language. The program can be a background program, a processor for the background, or a real-time foreground program.

A subsequent !XEQ command causes the RBM subtask S:LOAD to load the core image of the root segment (segment number 0) from OV into core storage. Subsequent segments (1 - n) are loaded by the root through the use of M:SEGLD.

When an !ABS control command is encountered, the Absolute Loader reads the absolute deck that follows (terminated by an !EOD) from the AI device and writes the core image copy onto the file to which the OV operational label is currently assigned. If OV has not been assigned, it will be assigned by default to the RBMOV file on the RAD. The program can be executed from a permanent SP (system processor) or UP (user processor) file either by inputting a "!name" command (where "name" is the name of the file on which the program was written), or an !XEQ command.

If a multisegment program is loaded, the Absolute Loader creates an OV:LOAD table at the end of the root. The root must always be the first load module and each succeeding load module is assigned a consecutive segment identification number, with the first succeeding segment starting at "1". In the OV:LOAD table, each segment's load address will be at its origin location and its entry address will be the transfer address generated by the END card image.

The form of the !ABS control command is

!ABS [size][,oplb$_1$][,oplb$_2$]. . .[,oplb$_n$]

where

size is an optional parameter for background programs only. It specifies the temp stack size required for the background program being loaded. If size is omitted, a temp stack size equal to the maximum size needed for all Monitor service routines (80) will be used. The temp stack will always be allocated at the start of background, and it is the user's responsibility to origin his program above the temp stack. For foreground programs, the size parameter is ignored and the temp stack pointers must be assembled as part of the program (i.e., in the TCB).

oplb$_1$,oplb$_2$. . . are operational labels used by the program that requires blocking buffers (i.e.,those labels that may be assigned to blocked RAD files). A maximum of 10 operational labels may be specified. When the program is loaded from the RAD for execution, the Monitor will ensure that enough blocking buffers are available for these specified labels assigned to blocked files.

Programs loaded under the Absolute Loader are subject to the following restrictions:

- No external references are permitted.
- The program must be in absolute form.
- Relocatable code may not be imbedded.

ASSIGN The !ASSIGN control command causes either a new or standard operational label to be equated with a specified (or temporary) file number. Since operational labels for the background are reset to the standard values at the beginning of a job by the Job Control Processor, an operational label assignment is in effect only until the next !JOB command is encountered or until it is again reassigned.

An operational label is a two-character name that is used as a label in referring to a device-file number. The convention of operational labels is used for the processors or any other program to make them device-independent, and also to give some mnemonic value to the input/output operations associated with the processors.

Device-file numbers are a logical means of referring both to a physical peripheral device and to a collection of information about that device; that is, the current file of information. Device file numbers are defined sequentially in the DEVICE FILE INFO parameter during SYSGEN.

Standard operational labels can be reassigned to different device-file numbers during SYSGEN or through !ASSIGN and !DEFINE control commands. Two tables of operational labels are maintained by the system; one is used for background (see Table 2) and the other for foreground. Device unit numbers (see Table 3) are also stored in the same two tables in the form of binary integer values.

Table 2. Standard Background Operational Labels

Operational Label	Explanation of Reference	I/O Device
AI	ABS binary input	CR,PT,MT,RD
BI	Binary input	CR,PT,MT,RD
BO	Binary output	CP,PT,MT,RD
CC	Control command input	KP,CR,PT,MT, RD
DO	Diagnostic output	Same as LO
GO[t]	Execution input (GO)	CR,MT,PT,RD
ID[t]	Debug ident file	RD
LI	Library input	Same as BI
LL	Listing log	Same as LO
LO	Listing output	LP,KP,MT,RD
OC	Operator's console	KP
OV[t]	Overlay (temporary)	RD
PI[tt]	Processor input	RD

Table 2. Standard Background Operational Labels (cont.)

Operational Label	Explanation of Reference	I/O Device
SI	Symbolic input	KP,CR,PT,MT, RD
S2[†]	Sigma 2/3 procedures	RD
UI	Update input	CR,PT,MT,RD
UO	Update output	PT,MT,RD
X1[†††]	Overlay Loader, Extended Symbol	MT,CR,RD
X2[†††]	Overlay Loader, Extended Symbol	RD
X3[†††]	Extended Symbol	RD
X4	Utility (verify)	RD,MT,CR,PT
X5[†††]	Utility (prestore)	RD

[†]These operational labels, if required by a processor, are automatically assigned to permanent files in the system data area by the Job Control Processor.

[††]The PI operational label is assigned to files in the System Processor and User Processor areas by the Job Control Processor.

[†††]These operational labels are automatically assigned to background temporary RAD files, with the file definition appropriate to the background processor being executed. These definitions are made from a table in the Job Control Processor that is selected by the first three characters of the processor name.

Table 3. Standard Device Unit Numbers

Device Unit Number	Standard Assignment
101	Keyboard/printer input
102	Keyboard/printer output
103	Paper tape reader
104	Paper tape punch
105	Card reader
106	Card punch
108	Line printer

The foreground operational labels reserved for use by RBM are as follows:

Label	Usage	Device
AL	Accounting log	RD
CK	Background checkpoint	
DP	Mount/remove key-ins	
EF	Error log	
ML	Program loading	
RM	Overlay input	

An assignment to file zero means that the operational label is not effective, and all references to this operational label result in a no-operation until it is reassigned. Note that some background processors (e.g., Utility) do not allow use of active operational labels assigned to file zero. See Appendix B for a complete description of operational label usage.

!ASSIGN commands can appear anywhere within the control command stack (except within a job step) and take effect immediately. That is, if the CC operational label is reassigned, the very next control command is read from the newly assigned device (unless the KP override has been imposed by an unsolicited key-in). The !ASSIGN command is used for both foreground and background operational labels. (The operator must key in FG before assigning a foreground operational label.)

There are four forms of the !ASSIGN command. Form 1 is

```
!ASSIGN oplb=device-file-number[,F][,(opt1)][,(opt2)]
```

where

oplb is either a two-character alphanumeric name in the foreground or background operational label table (or is to be placed in the table), or a FORTRAN device unit number, indicated by the prefix F: preceding the device unit number (see Table 3).

device-file-number is a decimal integer in the range 1 to 50.

F when present, declares that the assignment is to be included in the foreground operational label table. Otherwise, it is assumed to be in the background operational label table, and the file number must also be a background file number.

opt 1 and opt 2 are device specific options which may be one to four characters. If more than four characters are specified, only the first four will be used. Note that the device specific options are meaningful only for certain devices. Use of an unrecognized option for a device results in an error return of INVALID OPTION.

The following options are recognized for Model 3325/33 tape drives:

800	for 800BPI; NRZI recording
1600	for 1600BPI; phase encoded recording
ASCI[I]	for ASCII code conversion
EBCD[IC]	EBCDIC data (ASCII code conversion "off")

Form 2 of the !ASSIGN command is

```
!ASSIGN oplb=filename,area[,F][,S]
```

where

oplb is an operational label or a device unit number identified by the F: prefix.

filename is the name of an existing RAD file. The RAD file is rewound if it is blocked or compressed.

Only permanent RAD files can have a filename. Once the filename is entered in the dictionary by SYSGEN or RAD Editor, an !ASSIGN control command or call to M:ASSIGN can equate either an operational label or FORTRAN device unit number in this filename.

area specifies the area to search for the filename from the areas listed in Table 1.

F indicates that the assignment is to be included in the foreground operational label table.

S indicates that this file (if packed format) may use the sharable blocking buffer if provided by the Task Control Block.

F and S are not order dependent.

Form 3 of the !ASSIGN command is

```
!ASSIGN oplb=oplb[,F][,(opt1)][,(opt2)]
```

where

oplb is as defined above.

F if present, indicates that both operational labels are foreground; otherwise, both operational labels must be background labels.

opt1 and opt2 are as defined for form 1.

Form 4 of the !ASSIGN command is

```
!ASSIGN oplb = area,area[,F]
```

where

oplb is as defined above.

area identifies the disk area to which the oplb is to be assigned (must be specified twice).

F if present, indicates that the operational label is for the foreground; otherwise, it is assumed to be a background label.

This form of the ASSIGN command allows access to an area as if it were a file with the following characteristics:

Format:	random
Logical record size:	sector size in bytes
Write protection:	area write-protect code
BOT:	BOT of area
EOF:	none
EOT:	EOT of area

Examples:

Form 1:	!ASSIGN SI = 3
	!ASSIGN F:105 = 3
Form 2:	!ASSIGN OV = ROOT, UP
Form 3:	!ASSIGN LI = BI
Form 4:	!ASSIGN SI = CP, CP

ATTEND The !ATTEND control command indicates that RBM is to go into a wait condition on any abort from the background, and then read and process the next control command encountered when background processing continues after an unsolicited key-in. Its primary purpose is to offer improved recovery procedures. If an abort occurs without this control command being specified, JCP will reset the CC operational label to the standard value, skip all control commands, binary records, or data until it finds a new !JOB, !PURGE or !FIN command, and will not pause for operator intervention. In this "skip" mode, all EBCDIC records beginning with ! will be listed on the LL device, with an indication ('>' preceding the command) that they are ignored. This is the normal mode for closed-shop batch processing, without halts between jobs after aborts.

The form of the command is

```
!ATTEND
```

It exists for one job only, and usually immediately follows the !JOB command.

C: The !C: control command connects the designated real-time foreground task to a specified interrupt location, optionally armed and enabled as specified by the control code. The task may also be triggered by means of this connect operation if the code is equal to seven, providing that the task has previously been armed (i.e., with a previous !C: command, an !XEQ or "!name" command, or by a Q key-in).

The form of the !C: control command is

```
!C:  tcb[,code]
```

where

tcb is the address of the Task Control Block for this task. If the value is hexadecimal, it must be shown as +xxxx. If the Overlay Loader initializes the TCB by means of the TCB parameters, it does so completely, using load information and values on the TCB and BLOCK cards. No partial initialization of a TCB is allowed with the exception of the blocking buffer pool. If a user builds his own TCB, the TCB must begin at the execution location plus the "temp" value specified on the Overlay Loader !$ROOT command.

code when present, is the interrupt operation code. It overrides the initial TCB task code; a code of 7 triggers the task if it is armed.

Note: If "code" is not specified, the code given in the TCB will be used.

The !C: command does not change the contents of the TCB.

CC The !CC control command returns control to the currently assigned CC device and nullifies the effect of a previous KP key-in. The control command is honored regardless of whether or not the "skip" mode is in effect. The "skip" mode is cleared following this command. The form of the command is

```
!CC
```

DEFINE The !DEFINE control command allocates a portion of the background temporary RAD space for a specific operational label or device unit number by assigning the operational label to an unused device-file number, which in turn is linked to the specified portion of the RAD. Since temporary RAD files are not maintained by the Monitor, they have no name and are identifiable only by the operational label for which each file was created. The !DEFINE control command must precede the specific processor or user program to which it applies, since this temporary space is reset at the beginning of each job and at the subsequent reloading of the JCP (unless a !TEMP S control command is in effect). That is, the files are destroyed and the RAD space and all device-file numbers linked to it may be used by the next job.

The form of the !DEFINE control command is

```
!DEFINE oplb[,.per][,nrec],srec [,{R|U|C|B|P[,S]}]
```

where

oplb is an operational label or a FORTRAN device unit number (with a prefix of F:).

nrec is the number of logical records in the file.

.per indicates the percentage of remaining background temporary space to be allocated for this oplb.

srec is the logical record size, in bytes.

R defines the file as an unblocked random-access file.

U defines the file as an unblocked file.

C defines the file as a compressed EBCDIC file.

B defines the file as a blocked sequential file.

P defines the file as a blocked random-access file.

S flags the desire to use a shared blocking buffer if provided with the program task. It is meaningful only for packed (blocked random) files.

If neither R, P, U, B, nor C is specified, the file is defined as a blocked file (B). If R is input, srec is used as the granule size.

EOD Sections of data may be defined in a user's deck by inserting !EOD control commands at the end of each section. When an !EOD command is encountered, the Monitor returns an EOD status (when using the M:READ I/O routine). This is similar to a tape-mark on magnetic tape. Any number of !EOD control commands may be used in a job wherever required by the user or by a processor.

The form of the !EOD control command is

```
!EOD
```

FIN The !FIN control command specifies the end of a stack of jobs. When the !FIN control command is encountered, the Monitor writes it on the listing log to inform the operator that all current jobs have been completed and also writes !!BEGIN IDLE on the OC device. The Monitor then enters the idle state.

The form of the !FIN control command is

```
!FIN
```

FSKIP,FBACK,RSKIP,RBACK The file positioning control commands, !FSKIP and !FBACK, forward or backspace the specified device (magnetic tape or RAD file) immediately past the next file mark, or past the nth file mark if n files are specified (n = 1 for RAD files). !RSKIP and !RBACK perform similar functions but act on records rather than files. !RBACK and !RSKIP do not apply to compressed RAD files.

The forms of the control command are

```
{!FSKIP}
{!FBACK} device[,number][,F]
{!RSKIP}
{!RBACK}
```

where

device specifies the device to be positioned and is one of the following:

1. A device-file number, shown as a decimal integer.

2. A FORTRAN device unit number, shown as

   F:n

   where n is a decimal integer equal to the device unit number.

3. An operational label, shown as two alphanumeric bytes, the first of which is alphabetic.

number is the number of operations to be performed; if absent, one operation is assumed.

F indicates a foreground device/file. This indication is not required if a device-file number (DFN) is specified directly. Operations on a foreground device or file require that an FG key-in be in effect.

HEX The !HEX control command (SYSGEN optional) may be used to patch either the Monitor itself or any foreground program.

The form of the !HEX control command is

```
!HEX
```

The format of the patch record is described in Chapter 11 under "System Patching".

JOB The !JOB control command signals the beginning of a new job. The background operational labels and FORTRAN device unit numbers are set to their default assignments. All RAD temp files are closed.

This command always causes a page to be ejected on the LL device before the command is listed. The version of the RBM being utilized will be inserted following the last field on the !JOB command.

The form of the !JOB control command is

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

where

name has a limit of 12 characters.

account has a limit of six characters.

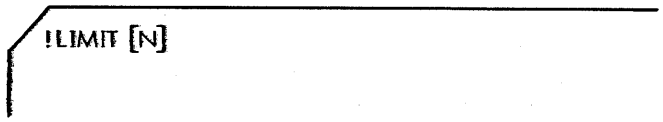
JOBC The !JOBC control command indicates a continuation of the current job. !JOBC closes all RAD temp files and resets all background operational labels to their default assignments (with the exception of "CC"). The !JOBC command does not clear the "attend" flag or the "skip" mode, nor does it terminate the effect of an FG or SY key-in. (A useful application of the !JOBC command is given in the Utility job deck example in Chapter 10.)

The form of the !JOBC control command is

```
!JOBC
```

LIMIT The !LIMIT control command (SYSGEN optional) is used to set a maximum on the execution time of a background program. This command is effective only if the job accounting option has been selected at SYSGEN. If the job exceeds the time limit, the job is aborted (TL) and is terminated with a postmortem dump (if that option was specified).

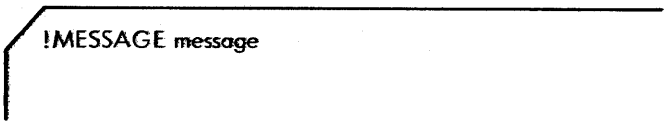
The form of the !LIMIT control command is



where N is the maximum allowable execution time in minutes (0 < N < 600).

MESSAGE The !MESSAGE control command is used to type a message to the operator. It is useful for messages concerning mounting tapes or setting certain device or Control Panel conditions. The command is listed on the OC device. There is no response.

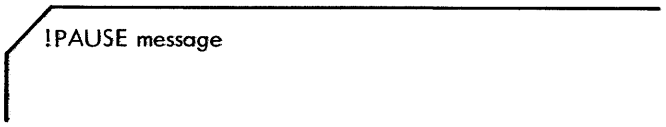
The form of the !MESSAGE control command is



where message is any comment to the operator, up to the full-card image size (total of 72 columns per card).

PAUSE The !PAUSE control command temporarily suspends background operation to allow the operator time to complete the job setup. Background operations resume when the operator performs an unsolicited S key-in. The command is listed on the OC device.

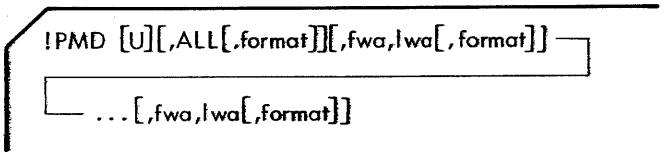
The form of the !PAUSE control command is



where message is a comment to the operator, up to the full-card image (total of 72 columns per card).

PMD The !PMD (postmortem dump) command causes the Monitor to dump the registers, plus selected areas of memory, at the end of a job step. The dumps are always onto the background DO device in specified format. The !PMD command is only effective for one job step.

The form of the !PMD command is



where

U indicates that PMD is to be entered regardless of the manner of background termination. Otherwise PMD is entered only if background terminates abnormally.

ALL indicates that all of background is to be dumped. If ALL is not specified and no other limits are specified, only the CPU registers are dumped.

fwa, lwa specifies the dump starting and ending locations. These values are hexadecimal if preceded with a plus (+) character.

format specifies the dump format as follows:

H	Hexadecimal (default, if format unspecified)
M	Mnemonic
I	Integer
E	EBCDIC

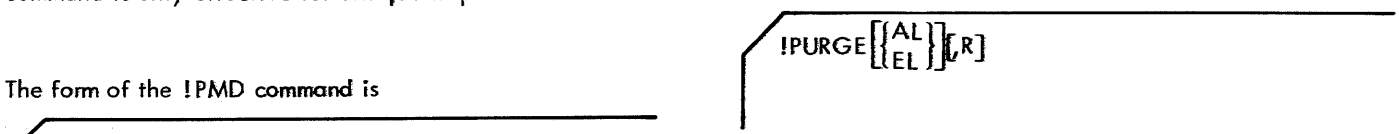
When a format of E is specified, each dump line will consist of hexadecimal values followed by EBCDIC translations, at the end of the line. Four limit pairs (fwa, lwa) may be specified. The CPU registers are always dumped, regardless of the limits.

An X (abort) key-in will terminate all postmortem dumps if performed while PMD is active.

PURGE The !PURGE control command (SYSGEN optional) is used to output the contents of either the job accounting file or error-log file, and optionally to reset (i.e., clear) the respective file. By use of the reset option and an assignment of the appropriate background operational label (see below) to a "hard copy" device (card punch, paper tape, or magnetic tape) a periodic off-line copy of the chosen file can be obtained and the corresponding RAD/disk space freed for further entries. (Operator messages will indicate the need for such action; in the error log case, a prompt response is necessary in order to prevent loss of records in this file.)

A !PURGE command will always be acknowledged whether in "skip" or "attend" mode.

The form of the !PURGE control command is



where

AL specifies the job accounting file (default).

EL specifies the error log.

R specifies that the indicated file is to be reset (i.e., cleared).

If neither AL nor EL is specified, AL is assumed. If R is specified, use of the command must be preceded by an (unsolicited) SY operator's key-in.

Accounting File Output. The contents of the accounting file are output, via background operational label LO, in the following format:

```
mm/dd/yy hhmm name account mmmm.mm
```

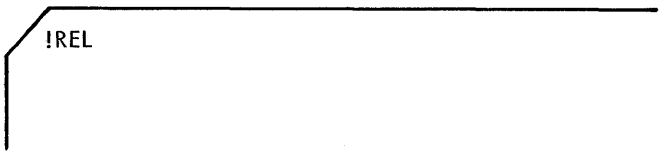
where mmmm.mm indicates job execution time to the nearest hundredth of a minute, e.g., 0003.85 minutes.

Error Log Output. The contents of the error log file are output via background operational labels DO and LO. The output via DO is an exact restorable copy of the error log, record by record, followed by two !EOD records. The output via LO is a readable representation of each record. If DO and LO are both assigned to the same device, the DO form of output is suppressed, i.e., LO predominates. If DO output is to be assigned to a magnetic tape containing previous log output, the recommended procedure is

```
!JOB
!PAUSE KEYIN SY,S
!ASSIGN DO = MT
!FSKIP DO
!FBACK DO
!PURGE EL,R
!UNLOAD DO
!FIN
```

REL Relocatable binary program modules to be loaded onto the GO file are preceded by an !REL control command The binary modules that follow must be in Xerox 16-bit Standard Object Language (see RBM/System Technical Manual 90 11 53). The modules may constitute a complete program, a root, or segments of a program. Checksum and sequence checks will be performed.

The form of the !REL control command is

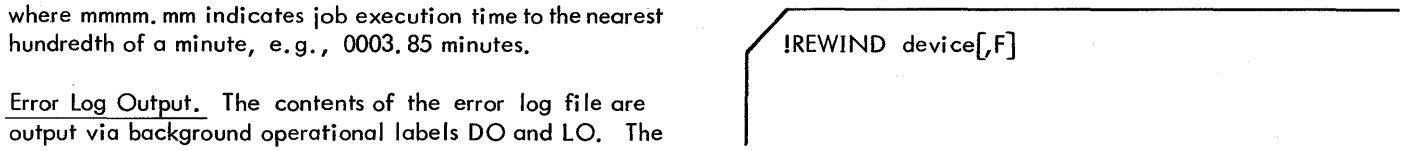


The modules are copied onto the file to which GO is currently assigned. If GO has not been assigned, it will be assigned by default to the RBMGO file on the RAD, which is rewound before the modules are copied. Several modules may be copied through the use of one !REL control command by stacking the modules. The final module must be followed by an !EOD control command that will cause the JCP to write an end-of-file (EOF) onto GO and then backspace one file. In this manner the GO file is positioned to accept additional input, but is always terminated by an EOF. The relocatable binary decks are loaded from operational label BI.

The !REL control command is a convenient method of obtaining additional hard copies of object modules produced on GO by Extended Symbol or FORTRAN. By assigning BI to GO and then reassigning GO to BO, modules will be copied from the original GO onto BO up to and including the EOF. BI should be rewound before each !REL command.

REWIND The !REWIND control command rewinds a magnetic tape or a RAD file and has no effect on other devices. The operation takes place immediately after the command is interpreted.

The form of the !REWIND control command is



where

device specifies (as in !FSKIP) the device to be rewound, by oplabel, fdun (FORTRAN device unit number), or DFN.

F indicates a foreground device/file. This indication is not required if a device-file number (DFN) is specified directly. Operations on a foreground device or file require that an FG key-in be in effect.

TEMP Normally, the temporary background space on the RAD is reset at the completion of each step within a job, so that a separate assembly and compilation can each have full access to this temporary area for scratch space as needed. The !TEMP control command is a means of altering this standard procedure. When used with the save (S) option, temporary files are not released after any job step within a job stack until either a !TEMP command is encountered with a reset (R) option or the next !JOB, !JOBC, or !FIN command is encountered.

The form of the !TEMP control command is

where either S or R is required

S means to save RAD temporary files between job steps within a job (e.g., between an assembly and a concordance).

R means to reset the RAD temp files after each job step.

T means truncate the previous file so that it will only be as long as the end-of-file. If no EOF has been written the shortened file will be one record long. Space recovered in this fashion can be reallocated by subsequent use of the !DEFINE command.

UNLOAD The !UNLOAD control command causes a specified magnetic tape or RAD file to be rewound in manual mode. Operator intervention is required to use the device again. If the device is a RAD file, the file is rewound to BOT and released by a call to M:CLOSE.

The form of the !UNLOAD control command is

```
!UNLOAD device[,F]
```

where

device specifies (as in !FSKIP) the file to be rewound off-line.

F indicates a foreground device/file. This indication is not required if a device-file number (DFN) is specified directly. Operations on a foreground device or file require that an FG key-in be in effect.

WEOF The !WEOF command writes the appropriate end-of-file mark on the output device. For magnetic tape, it is a tape mark; for the card punch or paper tape punch, it is an !EOD command; and for RAD files, it is a logical file mark.

The form of the !WEOF control command is

```
!WEOF device[,number][,F]
```

where

device specifies (as in !FSKIP) the device that is to have an end-of-file written on it.

number is the number of end-of-files to be written. If absent, one end-of-file is written.

F indicates a foreground device/file. This indication is not required if a device-file number (DFN) is specified directly. Operations on a foreground device or file require that an FG key-in be in effect.

XEQ The !XEQ control command loads the root module from whatever file the OV operational label is currently assigned to. For foreground programs, the command must be preceded by an FG key-in.

The form of the !XEQ command is

```
!XEQ
```

XED The !XED control command performs the same operations as the !XEQ control command except that !XED transfers control to RBM Debug through the entry point D:KEY when the root segment has been loaded. The message !!DKEY-IN will appear on the keyboard/printer and the user can then input Debug control commands. (See Chapter 12 for a discussion of RBM Debug.) The !XED control command causes the background operational label ID to be default-assigned to the RBMID file on the RAD if it is not already assigned.

The form of the !XED control command is

```
!XED
```

PROCESSOR CONTROL COMMANDS

Processors in the System Processor area and any user background or foreground program residing in the User Processor, Foreground or Background Program areas can be called by a processor control command. The commands have the format

```
!processor parameters
```

where

processor is the file name of a processor that must be distinguishable in the first three characters from system control commands (see Table 4). The order of search (by area) is SP, UP, FP, BP.

parameters are optional parameters interpreted by each particular processor.

Table 4. RBM System Processors

Name[t]	Description
FORTRAN	FORTRAN IV Compiler
RPG	Report Program Generator
OLOAD	Overlay Loader Subsystem
UTILITY	Utility Subsystem
XSYMBOL	Extended Symbol Assembler
RADEDIT	RAD Editor Subsystem
SORT	Sort Processor

[t]The RBM System Processor names are entered into the System Processor area dictionary with the RAD Editor !#ADD command. If the file name is less than eight characters, the name on the processor control command must exactly match the file name. If the file name is eight characters (maximum), the first eight characters of the name on the processor control command must exactly match the file name. Trailing nonblank characters beyond the eighth character in the processor control command name are ignored.

When a processor control command is read and interpreted by the Job Control Processor, the root segment of the specified subsystem is loaded from the RAD into memory. The JCP will assign all permanent RAD files used by the specified processor before the processor is executed unless these files were previously assigned via !ASSIGN commands. The JCP will also define all temporary operational labels used by the processor (by defining them as background temp files) unless they are previously defined via !DEFINE commands. JCP then transfers control to the processor.

When a requested processor is read into the background and attains control, this marks the beginning of job step. An example of a job stack illustrating its breakdown by job step is shown in Figure 2.

EXTENDED SYMBOL CONTROL COMMAND FORMAT

The Job Control Processor reads and interprets the !XSYMBOL control command and loads the Extended Symbol assembler from the RAD into background memory. The assembler continues to assemble programs until it encounters an end-of-file. The Extended Symbol assembler is called into operation with the command

!XSYMBOL [option$_1$, option$_2$, . . . , option$_n$]

where option can be

BA specifies batch assembly mode. XSYMBOL will ignore single end-of-files and will terminate only when two consecutive end-of-files are encountered.

BO specifies binary output.

CR specifies cross-reference listing.

DW specifies display warnings.

GO specifies output GO file.

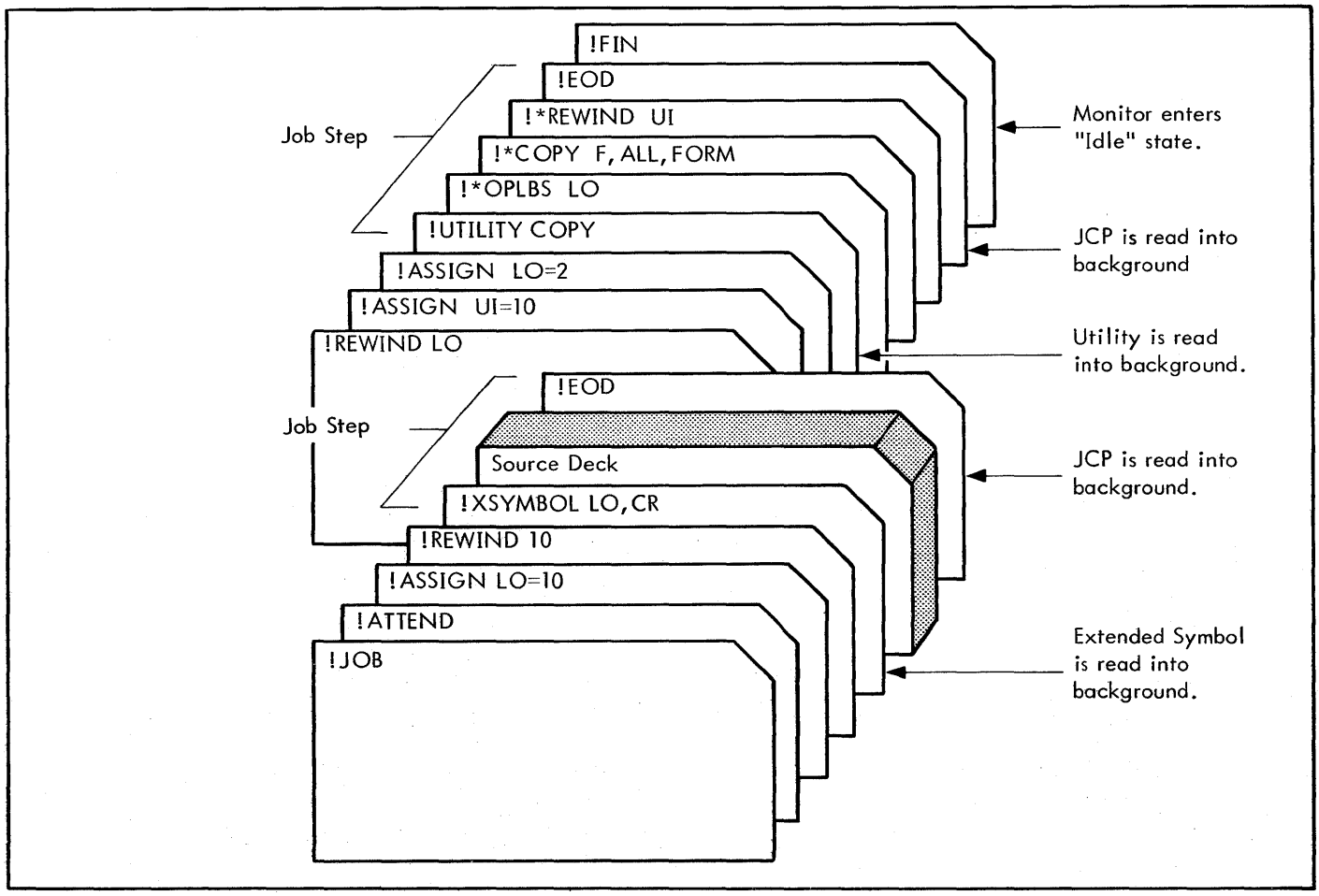


Figure 2. Job Stack Example

LO specifies list assembly output.

LU specifies list update.

NP specifies no standard procedure input.

PP specifies punch standard procedure file.

SL specifies simple literals.

SO specifies source output.

SS specifies symbol summaries.

UI specifies update input.

Any number of options may be specified and in any order. If no options are specified, the following options are assumed by default:

BO, GO, LO

The presence of any nondefault option requires that any desired default options (except SI which is always defaulted) must also be present.

FORTRAN IV CONTROL COMMAND FORMAT

The Job Control Processor reads and interprets the !FORTRAN control command and loads the FORTRAN IV compiler from the RAD into background memory. The compiler is called into operation with the command

!FORTRAN $s_1,s_2,\ldots,s_n$

where s_i can be

LO specifies an object listing.

LL specifies an object listing with data chains.

XP specifies extended precision real data instead of standard precision.

ALL specifies that multiple files are compiled. FORTRAN will ignore single end-of-files and will terminate compilation only when two consecutive end-of-files are read.

(The processor that is loaded may be either the Basic or ANS FORTRAN IV compiler, at the installation's option.)

Binary output is normally output on both the BO and GO devices. To suppress the BO or GO output, the user must assign the pertinent operational labels to 0 (see !ASSIGN and !DEFINE control commands in this chapter).

If no specifications are present, binary output on the BO and GO devices, a source listing, and standard precision mode are assumed by default.

RBM/PROCESSOR INTERFACE

Ground rules common to all system processors are:

- All processors operate in the background.
- With the exception of the UTILITY program, processors must use standard background operational label table assignments for their I/O requests. (See Table 2 for the standard background operational labels.)
- The first character of each line of the listed output from the processors is always interpreted as a vertical format character (carriage control) and is never printed. The RBM I/O routines treat the vertical format properly for the keyboard/printer, line printer, and magnetic tape.
- When the RBM transfers control to a background processor, the X register contains the address of the control card image, providing access to any parameters.
- At the completion of an assembly or compilation, the processor writes two end-of-files on the LO device, and then backspaces the LO device one file. The M:CTRL routine will treat these operations for the devices as described in the I/O section. This permits file processing of output on magnetic tape, if LO is assigned to magnetic tape. The processor writes an EOF on BO and GO at completion and then backspaces one file (GO and BO are separate options).
- The processor generally returns control to RBM by means of a call to M:TERM. RBM will immediately read from CC and if there is another control command for the current processor, it will reload the processor from the RAD.
- If overlay loading is required, the processor uses M:SEGLD. The overlay operational label for the background is PI.
- If an irrecoverable error occurs, the processor exits to RBM with a call to M:ABORT and displays the abort code in the X register and the abort location in the A register.
- Since all standard RAD files are defined by the Job Control Processor, the processors need not call M:DEFINE, but must call M:CLOSE to release blocking buffers in those cases where several RAD files are used but are not all open at one time.
- The first output line to LO from an assembly or compilation should contain a top-of-form format code.

GO AND OV FILES

Figure 3 shows how the JCP and Extended Symbol or Basic FORTRAN IV use the operational labels GO and OV. The

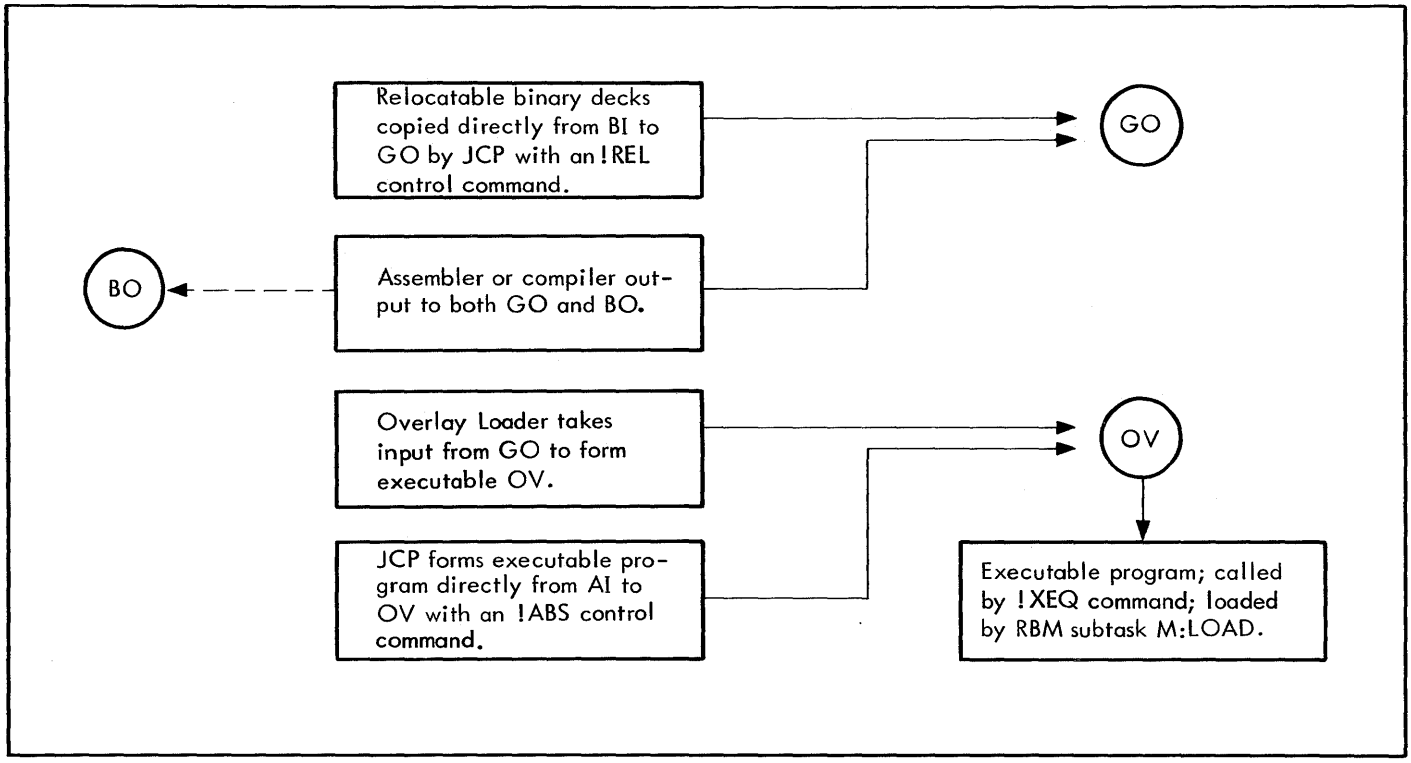


Figure 3. Use of GO and OV Files

GO and OV files are the files to which these operational labels are assigned by the JCP and are standard default files when no operational labels are specified. The GO file is a blocked, sequential file that contains relocatable binary decks read from the job stack, and binary ouput produced as a result of an assembly or compilation. After each module is loaded onto the file, an end-of-file mark is written and a backspace file is performed. Thus, at any point within a job stack the GO file contains all modules that have been loaded and is in position to accept others.

The Overlay Loader may now use the contents of the GO file to create an executable core image program and save this program on the random-access OV file. Absolute binary decks produced by an assembly may also be written (in executable core image form) onto the OV file by JCP through use of the !ABS command.

3. OPERATOR COMMUNICATION

SYSTEM COMMUNICATION

When events take place in the system that require operator intervention, or when one job is completed and another job begins, RBM informs the operator of these conditions by messages on the keyboard/printer. All such messages from the Monitor begin with two exclamation marks (!!)and are described in Table 5.

Generally, these messages require no operator response on the keyboard/printer but may indicate that some peripheral device needs attention. In some cases, the operator must interrupt and key in a response after correcting the specified problem.

I/O RECOVERY PROCEDURE

If a message concerns an I/O error condition, the Monitor I/O routines that generated the message will be waiting to sense a change of state in the device. (A change of state is defined as a change from manual to automatic, or from automatic to manual and back to automatic, depending on the initial condition.) When the change of state is sensed, the operation is retried. Thus, if the device is EMPTY, it need only be placed in the automatic mode. If there is a PUNCHES error or a FAULT on the card reader, the reader is unloaded, the bad card is corrected and replaced, and the reader is returned to the automatic mode.

Table 5. Monitor Messages

Message	Meaning
!!AL IO ERROR[†] !!BEGIN WAIT	An irrecoverable I/O error has occurred while accessing the accounting file, normally because of a hardware failure or unavailability of operational label AL. The correct assignment of this operational label is to RBMAL, SD. An attempt should be made to recover the contents of the accounting file as stated above. If this recovery fails, the operator may gain control through a KP key-in and then an FG key-in to allow foreground modifications; the foreground operational label AL may then be reassigned (e.g., !ASSIGN AL = RBMAL, SD, F or !ASSIGN AL = 0,F). Note: Assignment of the foreground operational label AL to zero will inhibit the logging of job stack entries into the accounting file.
!!AL OVERFLOW[†] !!BEGIN WAIT	The accounting file (RBMAL) cannot accept another entry. The accounting file is allocated at SYSGEN and accommodates 74 entries. (The user may increase or decrease this capacity via the RAD Editor.) At this point, normal error recovery will be a key-in of KP to gain keyboard/printer control. Next, a key-in of SY will permit access to the accounting file. The operator should now assign the background operational label LO to a hardcopy device (e.g., paper tape, card punch). Input of a !PURGE control command specifying the clear option (i.e., !PURGE AL,R) causes the contents of the accounting file to be copied onto that device and clears the accounting file. The job stack causing the overflow can now be reentered.
!!ATTEND ERROR xx	JCP has read an erroneous control command while operating in the ATTEND mode, in which case RBM goes into a wait state after typing this message. After a subsequent S key-in, RBM will process the next control command.
!!BEGIN IDLE	JCP has just read a !FIN card (which completes a job stack) and background has gone into an idle state. Processing will resume on a new job stack following an unsolicited S key-in.
!!BEGIN WAIT	The background has executed a WAIT request. An unsolicited S key-in will continue background processing.
!!BKG CKPT	Background has been checkpointed as a result of a foreground program request.

[†]This alarm occurs only if the RBM job accounting option has been exercised at SYSGEN.

Table 5. Monitor Messages (cont.)

Message	Meaning
!!RELEASE, dtnn	The specified device has been released for background use.
!!BKG RESTART	Background has been restarted from its point of interruption.
!!BKGD xx ABORT, LOC yyyy	The background job has aborted at location yyyy for the reason specified by abort code xx. If the Job Control Processor initiated the abort, a detailed explanation will be written on the background DO device. If the system is operating in the "attend" mode (see !ATTEND), RBM will perform any required postmortem dumps and then go into a wait state after an abort. After a subsequent S key-in, RBM will attempt to process the next control command from the CC device. If the system is not operating in the "attend" mode, RBM will not go into the wait state but will perform any required postmortem dumps and immediately begin reading from the CC device. All data cards and control commands will be skipped until a !JOB, !PAUSE, or !FIN card is found. Only a !JOB card will clear the "skip" mode. All control commands are listed on the LL device with an indication (> character) preceding the command to show that they are being ignored.
!!JCP	JCP has begun to read control commands. This message occurs at the beginning of a job and between steps within a job (e.g., when an assembly is completed). If CC is assigned to the keyboard/printer (as a standard assignment, or after a KP key-in), the input light on the keyboard/printer will indicate that RBM is ready for input of a control command.
!!CC NOT ASSIGNED	JCP is unable to read a control command because the CC oplabel either is assigned to DFN 0 or was not assigned during SYSGEN.
!!SYSERR xx	The Monitor has encountered some condition that will not permit further operation or a foreground task has generated an abort condition (see "Machine Fault Task" subheading in Chapter 6 and the "C" bit in Table 19). xx may be any one of the following: OP Operator-initiated system halt. SP The RAD device containing RBM cannot be recognized. ET An EIOP timeout has occurred. A system reset is necessary to continue. PE A task has generated a memory parity error. MF A task has generated a machine fault (probably the result of incorrect Direct I/O). PF A power failure has occurred at a time when RBM cannot recover.
!!dtnn EMPTY	The device specified is in the manual mode and may be out of paper, cards, or tape.
!!dtnn ERROR [,TRK xxxx]	There has been a parity or transmission error on the device. If any automatic retries were specified, they will have been performed before this message is output. A CR device will indicate that an error card is in the output stacker. Recovery procedure is described above under "I/O Recovery Procedure". If dt is RD, xxxx will be the errored track number, which is determined from the remaining byte count.

Table 5. Monitor Messages (cont.)

Message	Meaning
!!dtnn FAULT	Some condition on device type dt with physical device number nn (hexadecimal) has caused this device to become nonoperational. The recovery procedure is described above (in the discussion under change of state). The operation is automatically retried when the device goes into the automatic mode; it is neither necessary nor possible for the operator to type in a response.
!!dtnn PUNCHES	An invalid punch combination has been sensed on an EBCDIC image. The card will be stacked in the alternate stacker (if there is one).
!!dtnn DATA RATE	A data rate overrun has occurred. If any automatic retries were specified, they will be performed after this message is output.
!!dtnn UNRECOG	Device type dt with device number nn (hexadecimal) is not recognized by the I/O routines. If the device is a magnetic tape unit, the requested drive may not be dialed in properly or power may be off in either the unit or the controller.
!!dtnn WRT PROT	The RAD or magnetic tape is physically write-protected. If a RAD file is logically write-protected, this message will not appear but appropriate status will be returned.
!!dil REQUEST,dtnn	A request has been made to reserve the specified device. The operator should prepare the device and then reserve it through use of the FR key-in. dil refers to the dedicated interrupt location of the requesting task.
!!dil RESERVE,dtnn	The specified device has been reserved for foreground use for the task whose dedicated interrupt location is dil.
!!FRGD xx ABORT, LOC yyyy TCB zzzz	The foreground task with a TCB at location zzzz has aborted at location yyyy for the reason specified by abort code xx. The corresponding interrupt level will be disabled and if the task occupied nonresident foreground, an unload operation will be initiated. Background processing will continue. Because this message is written at the monitor priority level, only the abort message for one foreground task (the lower priority level task) will appear if two foreground tasks abort consecutively.
!!KEY ERROR[,comments]	The monitor could not process an unsolicited key-in response. The message usually indicates a format error on the key-in, where comments may be one of the following: NO AR — The wrong disk pack was mounted for an M key-in and the area could not be found. DEVICE — One of the following conditions was detected: 1. This device was not defined, 2. The device does not have removable areas. Applies to M and R key-ins. NO BTL — There is no bad track list for the device specified. 2 IO ERR — The device specified in the 'M' key-in cannot be correctly accessed.

Table 5. Monitor Messages (cont.)

Message	Meaning
!!KEY ERROR[,comments] (cont.)	2 ERR n — The following error codes are defined in the 'M' key-in processing: n = 1 The expected device number parameter is not two characters. = 2 The key-in exceeds the 20 characters maximum. = 3 The field exceeds the maximum length of eight characters. = 4 During the 'all' option, an area is defined on another device. = 5 The area specified is not found on the device. = 6 The area name specified is found on another device. = 7 An expected area name is not two characters. = 8 Sector 2 does not contain a bad track list. = 9 No bad track list for this device is found in the system tables. = 10 An option other than the 'all' or area option is specified. = 11 There is no room available in the Master Directory for the specified area. IN USE — If the key-in was an M (mount), the area must be removed. If the key-in was R (remove), files must be closed in the area (perhaps by an abort or unload). OVFLOW — The Master Directory table length will not allow this key-in to be processed. DFN/OP — The Device File table or Operational Label table has overflowed. IO ERR — The device specified in the 'M' key-in cannot be correctly accessed. TEMP STACK — The RBM Temp Stack has overflowed.
!!MESSAGE comments	A !MESSAGE control command has been read. The comments field may contain tape mounting or other instructions. RBM continues to read from the CC device after the message is typed out.
!!PAUSE comments	A !PAUSE control card has been read. The comments field may contain tape mounting information or other instructions. A control panel interrupt followed by an S key-in will cause RBM to continue reading from the job stack.

Table 5. Monitor Messages (cont.)

Message	Meaning
!!NO 'RBMPMD' FILE OR DFN	A portion of background could not be saved. The first part of background will be dumped as zeros.
!!POWER ON	The system has experienced a power failure and the power-fail-safe option has been implemented. If the computer is a Sigma 2 or is a Sigma 3 with no external interrupt and no critical foreground tasks, or if the background or RBM Control Task was active, execution will continue; otherwise it will crash. If the latter case, the operator should reboot RBM from the RAD and restart the background.
!!dtnn NOISE REC	A noise record has been detected on magnetic tape and ignored. (A noise record is one that contains less than eight bytes and an irrecoverable parity error).
!!dtnn BAD TAPE	The magnetic tape mounted on device dtnn contains a bad spot that cannot be skipped when writing. The operator should mount a new tape and (if possible) rerun the job.
!!ENTER DATE AS MM/DD/YY	A program request was made via M:DATIME for the date specifying that the operator be unconditionally solicited for the date.
!!ENTER TIME AS HR,MN	A program request was made via M:DATIME for the time specifying that the operator be unconditionally solicited for the time of day.
!!ERRFILE OVERFLOW IMMINENT	The Error Log is about to overflow. Log entries will soon be lost unless the operator performs a !PURGE EL,R operation (see the !PURGE control command).
!!ERRFILE OVERFLOW,PURGE	The Error Log has overflowed and log entries are being lost. The operator must perform a !PURGE EL,R as soon as possible (see the !PURGE control command).

OPERATOR CONTROL

Operator control of RBM is achieved by one of two methods: solicited or unsolicited.

SOLICITED CONTROL

Solicited control will normally be in the form of a specific request from a foreground or background program and should always be directed to the operational label OC – Operator Console. There is no standard format for the response to a solicited control.

UNSOLICITED CONTROL

All forms of unsolicited control are initiated when the operator activates the INTERRUPT switch on the Processor Control Panel. Unsolicited control may take one of two forms:

1. An unsolicited key-in request.
2. A forced foreground disable.

The active foreground task will be disabled and a call will be made to M:EXIT if all of the following conditions are true; otherwise, a key-in response will be requested:

1. The value in the data switches has changed since the last activation of the Control Panel Interrupt (or since boot).
2. The value in the data switches matches the address of the dedicated interrupt location of the current task, as specified in word 2 of the standard Task Control Block. See Table 19. Note that this implies that the active task must call M:SAVE.

   Conditions 1 and 2, when taken together, simply mean that the operator must intentionally enter the appropriate value in the data switches; an accidental disable cannot normally occur.
3. The active foreground task (that is, the one to be terminated) must have a hardware priority lower than the Control Panel Interrupt level.

If a forced foreground disable is specified, a foreground abort message will be written; otherwise the Control Panel Interrupt Task sets a flag in the RBM Control Task status word and triggers RBM. The Control Panel Interrupt Task then exits.

When the RBM Control Task becomes the highest priority task in the system (that is, when all real-time foreground tasks are nonactive), it issues an output message

!!KEY-IN

and requests input (up to 20 characters) from the operator. Because of possible delays associated with messages to and from the operator, no devices used for time critical operations should time-share an I/O channel used for operator communications. Each key-in must be terminated with the New Line (NL) code. The backspace (← or control-X) and delete (EOM or control-H) codes may be used before the New Line is typed to correct a mistyped key-in. The analysis and subsequent action from the unsolicited key-in is performed at the RBM Control Task priority level. Each key-in mnemonic must be followed by a space before its argument list.

Specific key-in responses under RBM are:

*** comment** Insert a comment. Useful for remote assist dialog. Note that a blank must follow the asterisk.

BL oplb=dfn[,P] Permits change of operational label assignments during running of background programs.

where

oplb is an assigned operational label or FORTRAN device unit number.

dfn is a decimal number specifying a legitimate device file number.

P is an optional permanent change of the default assignment until system reboot.

BL oplb=oplb[,P] Alternate version of BL (Background Label) key-in above.

BR[dt]nn Release the specified device for the next waiting task. The characters representing the device type are optional but, if input, will be used to validate the request.

C: tcb[,code] Connect the specified real-time foreground task to the dedicated interrupt location.

where

tcb is the address of the task control block for this task. (If the value is hexadecimal, it must be shown as +xxxx.) If the Overlay Loader initializes the TCB by means of the tcb parameters, it does so completely, using load information and values on the TCB and BLOCK cards. No partial initialization of a TCB is allowed with the exception of the blocking buffer pool. If a user builds his own TCB, the TCB must begin at the execution location plus the "temp" value specified on the Loader !$ROOT command.

code if present, overrides the initial code in the TCB for the task; a code of seven would cause the level to be triggered. If code is not present, it will be derived from the task control block.

CC Remove the keyboard/printer override of the CC device. The next control command will be read from the background operational label CC. This operator key-in is identical to the CC control command.

DA nn Make available a device that was previously declared unavailable (i.e., "down"), where nn is the address of the device.

DB[†] xxxx,yyyy Dump locations xxxx to yyyy if requested; otherwise, immediately dump all of background memory on background device DO. This key-in can be input at any time for debugging purposes. The dump will be in hexadecimal.

DC[†] [**CHAN,chan** / **DEV,dev** / **DFN,dfn** / **OPLB,** { **fdun** / **oplb** } [**,F** / **,B**]]

Display the I/O-error and I/O-access counters for either one or all channels, as specified by the form of key-in.

where

chan is a one- or two-digit hexadecimal channel number. The limits on chan are $0 \leq chan \leq 1B$.

dev is a two-digit hexadecimal device address.

dfn is a one- or two-digit device file number (DFN), in hexadecimal.

fdun is a FORTRAN device unit number. If the second parameter begins with "F:" or a numeral, an fdun is assumed.

oplb is a two-character operational label.

[F / B] if present, indicates that the specified operational label or FORTRAN device unit number is for the foreground (F) or background (B). If not specified, background is assumed.

If no parameter is specified, all channel error and access counters are displayed. (All channel and device numbers specified must have been declared at SYSGEN time.)

[†] SYSGEN optional.

The format of the display message output in response to a DC key-in is as follows:

CHAN cc ERRORS eeee ACCESSES aaaaaaaa

All values are displayed in hexadecimal and reflect the number of errors and accesses since the last counter reset (see the RC key-in) or since system boot, whichever is more recent.

DE[†] Causes Debug (if Debug is part of the system) to request the input from the keyboard/printer.

DF[†] **xxxx,yyyy** Dump locations xxxx to yyyy if requested; otherwise, dump all of foreground on background device DO. The dump will be in hexadecimal.

DM[†] **xxxx,yyyy** Dump locations xxxx to yyyy if requested; otherwise, immediately dump all of RBM on background device DO. The dump will be in hexadecimal.

D[T][†]**mm/dd[/yy[,hrmn]]** Reset the calendar date within RBM and continue processing if the Monitor is in an idle or wait state.

D[T][†]**mm,dd[,yy[,hr,mn]]** Alternate version of D[T](Date) key-in above.

DR[†] **[dn] xxxx,yyyy** Perform a selective dump of the RAD device dn to background device DO, where xxxx and yyyy are the first and last sectors of the block of sectors to be dumped. If dn is omitted, the RAD containing the SP area will be dumped. If dn refers to an undefined or non-RAD device, an error message will be written. If a consecutive series of sectors are all zeros, they will be skipped unless the last sector of this zero series is yyyy, in which case it will be dumped. For example, if "DR 100,200" is keyed in, and sectors X'1B0' through 'X'215' contain zeros, X'100' through X'1AF' and sector X'200' will be dumped. This key-in applies only to the 7202, 7203, and 7204 RADs.

The RAD dump routine performs RAD input with interrupts inhibited, and therefore should not be used when response time is critical.

DS nn,mm[,dfn] Substitute one device for another, i.e., change the device address associated with one or more device/file numbers (DFNs). This key-in is used mainly for reassigning Model 7332/33 (1600 BPI) magnetic-tape device address when one of these units has been declared unavailable. In the key-in syntax, nn is the hexadecimal device address to be replaced by mm, and dfn (optional) is the single DFN for which the substitution is to be made. (The dfn is checked to ensure correspondence to nn prior to change.) If dfn is not specified, all DFNs that currently point to device nn will be so modified. The message

!!CHANGED DFN dfn

will be issued for each DFN so modified, so that one or more can later be changed back to its original assignment. The specified devices (nn and/or mm) may be either available or unavailable when the key-in is made. The availability status of the mm device is applied to all DFNs reassigned to it.

This key-in does not apply to disk/RAD devices, nor may it be used to substitute one device type for another (e.g., Model 7322/23 for Model 7332/33, or tape for printer).

DU nn Declare a peripheral device unavailable (i.e., "down"), where nn is the device address. This key-in is not valid for the system RAD or disk, nor for the operator's console. Subsequent M:READ, M:WRITE, M:CTRL, or M:IOEX references to the "down" device will return a device-unavailable status.

F[†]**[oplb[,F] / fdun[,F] / dfn]** Dump the information described below for the specified file, or dump the operational label table only.

where

oplb is an operational label that indirectly specifies the desired DFN. F indicates a foreground operational label.

fdun is a FORTRAN Device Unit Number (e.g., F:101) that indirectly specifies the desired DFN. F indicates a foreground fdun.

dfn is a Device File Number (DFN).

If no parameter is specified, only the operational label table will be displayed.

When a parameter is specified, the following information will be output on background DO device for the desired DFN in addition to the operational label table.

- Contents of the specified Device Channel Status Tables.
- Contents of the specified File Control Tables.
- Contents of the specified I/O Control Tables.

If the file is a RAD file, the following additional information will be output:

- Contents of the specified I/O Control Sub-table.
- Contents of the blocking buffer assigned to the specified file, if one exists.

[†]SYSGEN optional.

FG[,S] Must precede any job stack operation affecting the foreground or the operation will be aborted. This key-in is effective until the next !FIN or !JOB command is encountered. Since the key-in is normally input in response to a !PAUSE command, the optional S key-in will clear the wait state.

FL oplb=dfn[,P] Permits foreground operational label assignment changes during system operation. The changes will be reset to SYSGEN values upon system reboot.

where

oplb is an assigned operational label or FORTRAN device unit number.

dfn is a decimal number specifying a legal device file number.

P is an optional permanent change until system reboot.

FL oplb=oplb[,P] Alternate version of FL oplb=dfn[,P]

FR [dt]nn Reserve the specified device for foreground use. The characters representing the device type are optional but, if input, will be used to validate the request. The device type will be required to distinguish PT40 from KP40, etc.

H[t] Input hexadecimal patch cards from background device CC. (See Chapter 11 for the format of the patch cards.) Patches to RBM or foreground must be preceded by an SY or FG key-in.

KP Begin reading control commands from the keyboard/printer. The key-in goes into effect immediately and stays in effect until a CC key-in or !CC control command is encountered.

L message Enter a message into the system's error log. The message may consist of up to 18 characters; it will be truncated to that length if necessary. If error logging was not specified at SYSGEN, this key-in will result in a KEY ERROR message.

M dn[,[vsn][,ar$_1$,ar$_2$,...,ar$_n$]][tt] Mount areas "ar" on device "dn". The operator must mount the disk pack containing areas "ar$_i$" on device "dn" before making this key-in. Unless the area specified is Xn, the disk pack will be read to ensure that it contains the specified areas. If no areas are specified, then all areas on the disk pack will be added to the Master Directory in core, otherwise, only the areas specified will be added to the Master Directory. If the Master Directory already contains an entry for an area, an error message !!KEY ERROR, IN USE will be output. The currently mounted area must be removed with an R (remove) key-in and the M (mount) key-in reissued. Other error messages are listed in Table 6. The optional vsn parameter is a three- to eight-character volume serial number.

For cartridge disks which contain a bad track list (Models 7251/52 and 3231/32/33), the M keyin will read the bad track list into the system tables.

Twenty characters, including ⓃⓁ, is the maximum that can be input for any one key-in. If an M key-in exceeds 20 characters, it can be divided into two parts. For example

M dn,67890123,a1,a2,a3ⓃⓁ

is 23 characters long. It may be divided up as

M dn,67890123,a1 ⓃⓁ followed by

M dn,a2,a3 ⓃⓁ

M dn,Xn[,wp] Mount area Xn on device "dn".

where

wp specifies the write-protection level for the area as denoted by one of the following codes:

Codes	Write-Protection Level
NO (or N)	No write-protection; background or foreground programs may write on the file.
BG (or B)	Write permitted by background programs only.
FG (or F)	Write permitted by foreground programs only. Background programs may write on the file if an SY keyin is in effect.
SY (or S)	Write permitted by RBM only. Foreground or background programs may write on the file if an SY keyin is in effect.

If the wp parameter is omitted, the default write-protection level is NO.

M dn,BTL Input the bad track list from device "dn" and move it into the system tables. No areas will be added to the Master Directory in core.

[t] SYSGEN optional.

[tt] Recognized only if a disk pack unit has been declared at system generation.

Q name Queue specified program for subsequent execution in nonresident foreground. As soon as this space is free, the requested program is loaded. If the queue stack is full or if the specified program is not found in the directory, an error message is output on the assigned foreground oplb, DO.

R dn[,ar_1,ar_2,...,ar_n][tt] Remove areas from the Master Directory. If no areas "ar_i" are listed, all areas on the device will be removed from the Master Directory. For the cartridge disks which contain a bad track list (Models 7251/52 and 3231/32/33), the bad track list is removed from the system tables. If any files are in use within the areas, removal does not occur and a !!KEY ERROR, IN USE message is output. An X (abort) keyin to abort a background program or an UL (force unload) keyin to unload a foreground program may overcome an IN USE situation for removal.

RA nn Xerox 530 systems only. Allow connection by dial-in of a remote-assistance terminal to the specified device number (that must be assigned to a Xerox Model 4194 or equivalent device). Following execution of this key-in, the remote-assistance capability is automatically involved upon detection of a ring indication on the data set for a specified device. This key-in is applicable only to data sets with an automatic answering feature; e.g., a Bell Series 103A or equivalent.

A foreground receiver (X'1B3') may be executed following completion of the remote connection.

RE dn Model 530 systems only. Allow connection of a remote assistance terminal to the specified device number (which must be assigned to a Xerox Model 4191, 4192, 4193, or 4194 terminal). Following execution of this key-in, the remote assistance terminal capability is invoked by completing a telephone connection with the data-set for the associated device.

RC[t] [**CHAN,**chan / **DEV,**dev / **DFN,**dfn / **OPLB,**{fdun / oplb}[,F / ,B]] Reset the I/O error and I/O access counters for either one or all channels, as specified by the form of the key-in.

where

chan is a one- or two-digit hexadecimal channel number. The limits on chan are $0 \leq chan \leq 1B$.

dev is a two-digit hexadecimal device address.

dfn is a one- or two-digit device file number (DFN) in hexadecimal.

fdun is a FORTRAN device unit number. If the second parameter begins with "F:" or a numeral, an fdun is assumed.

oplb is a two-character operational label.

[F / B] if present, indicates that the specified operational label or FORTRAN device unit number is for the foreground (F) or background (B). If not specified, background is assumed.

If no parameter is specified, all channel error and access counters are reset. (All channel and device numbers specified must have been declared at SYSGEN time.)

RD dn Model 530 systems only. Disconnect the remote assistance terminal from the specified device number (which must be assigned to a Xerox Model 4191, 4192, 4193, or 4194 terminal).

S Continue processing if Monitor is in an idle or wait state. If there is a waiting background program, continue processing that program. If there is no background program, begin reading control cards from the CC device. (Monitor can get into the wait state from a W key-in or !PAUSE command or into idle from a !FIN command.)

SY[,S] Permit modification of system files on the RAD to take place until the next !JOB or !FIN command is encountered. This key-in is a double check (similar to the FG key-in) to prevent accidental destruction of the RAD files. Since this key-in is normally input in response to a !PAUSE command, the optional S will clear the wait state.

T hrmn Reset the RBM system time, hour and minutes.

T hr,mn Alternate version of T hrmn.

TC nn Cause an I/O timeout to occur on the channel associated with hexadecimal device address nn. This keyin will initiate a retry of an I/O operation for a device which was formerly in need of operator intervention. In systems with Clock1 the retry will be automatic after 30 seconds but if Clock1 is excluded, the operator must perform this key-in. This key-in is not required if the device is in the "manual" condition, merely return the "automatic" condition and the I/O operation will complete.

UL Force an unload of the program occupying the nonresident foreground area. Note that operator key-ins can interrupt the background program at any time. Operator intervention cannot take place while there are active foreground programs, and will be delayed until they terminate.

W Background goes into a wait state.

X Abort the background job with any dumps requested, and output error code OP and a printed message showing the location of last background instruction executed. If the Postmortem Dump program is already active, it will be terminated.

Z Terminate the current background job including the Postmortem Dump program without performing postmortem dumps (abort code ER is output).

[tt] Recognized only if a disk pack unit has been declared at system generation.

4. MONITOR SERVICE ROUTINES

BRANCHING TO SERVICE ROUTINES

Under RBM, foreground and background programs may make calls on the Monitor to perform various services or privileged operations. (See Table 6.) For background requests, a branch to protected memory will trigger the protection routine which examines the branch for validity. If the protection violation is one of a permissible set of "controlled" violations, the branch is permitted; otherwise, the background job is aborted with a suitable error message giving the location to which the branch was attempted. If the branch is valid, the protection routine will permit the branch to the appropriate Monitor service routine.

All service routines are completely reentrant. Hence, they can be used by multiple tasks on a completely independent basis. Table 6 shows the routines requiring temporary space in the user's temp stack.

There are two different methods of executing a branch to one of these Monitor service routines: the conventional method is to declare the service routine name as an external reference and have the Overlay Loader satisfy the reference at load time. (In this case, the address literal will be in the user's program, and will be filled in by the Overlay Loader.) The other method is to branch indirectly through the address literal in the zero table (see Appendix A) using the absolute address given in Table 6. This is a useful technique for an absolute foreground program assembly, or for a processor or other programs that are self-relocating. It also requires less program space and may make it unnecessary to reload a permanent program following an update SYSGEN.

The B register is always saved and restored since it is used to point to temporary space. All other registers are volatile. The return address (specified by the L, T, or A register) must point to the background area if the routine is called (branched to) from the background. Otherwise, a protection violation abort occurs.

Table 6. Transfer Vector for Monitor Services

Address			F†/B/O		Words of Temp Required	
Dec.	Hex.	Routine		Purpose of this Routine	Min.	Max.
199	C7	M:FSAVE	F	M:SAVE Function if all registers previously Saved	0	0
200	C8	M:IOEX	O	Device-Dependent I/O Driver	16	16
201	C9	M:READ		Device-Independent Read Routine	19	51
202	CA	M:WRITE		Device-Independent Write Routine	19	51
203	CB	M:CTRL††		Device-Independent Control Routine	50	62
204	CC	M:DATIME††		Calendar Date and Time of Day	37	37
205	CD	M:TERM		Normal Termination of Background	0	0
206	CE	M:ABORT		Abnormal Termination of Background	0	0
207	CF	M:SAVE	F	Save Registers on Real-Time Interrupt	0	0
208	D0	M:EXIT	F	Restore Registers on Foreground Exit	0	0
209	D1	M:HEXIN		Hexadecimal to Integer Conversion	0	0
210	D2	M:INHEX		Integer to Hexadecimal Conversion	0	0
211	D3	M:CKREST	F	Checkpoint/Restart Background	0	65
212	D4	M:LOAD††		Load Nonresident Foreground or transfer control to another background task	32	32
213	D5	M:OPEN††		Open Blocking Buffer for RAD File	32	32
214	D6	M:CLOSE††		Close Blocking Buffer for RAD File	33	33
215	D7	M:DKEYS		Read Data Keys	0	0
216	D8	M:WAIT††	B	Execute Wait Loop from Background	34	66
217	D9	M:SEGLD		Load Overlay Segment	29	61
218	DA	M:DEFINE††	B	Define RAD Files in Background Temp Area	32	32

Table 6. Transfer Vector for Monitor Services (cont.)

Address			F[t]/B/O		Words of Temp Required	
Dec.	Hex.	Routine		Purpose of this Routine	Min.	Max.
219	DB	M:ASSIGN[tt]		Assign Operational Labels	37	51
220	DC	M:POP		Release Dynamic Temp Space	0	0
221	DD	M:RES		Reserve Dynamic Temp Space	0	0
222	DE	M:OPFILE		Convert Operational Label to Device-File Number	0	0
223	DF	M:RSVP[tt]	F/O	Reserve or Release Peripherals	39	71
224	E0	M:DOW[tt]	F/O	Diagnostic Output Routine and Error Logger	32	64
225	E1	M:COC[tt]	F/O	Communications Handler	44	44

[t]F = foreground only; B = background only; O = SYSGEN option.

[tt]These routines are nonresident RBM overlays. All nonresident RBM overlays require a minimum of 32 temp memory locations to load the routine.

Notes: 1. To branch to one of these routines, branch indirectly through the specified address above after RCPYI P,L (except M:RES which is called following an RCPYI P,T).

2. The minimum temp space required is the number used by the routine itself. The maximum temp space is the number required by this routine and those it calls, plus 19 if any of the routines are nonresident RBM overlays. For example, M:READ (19) may call Q:ROC to load M:OPEN (13) and Q:ROC may reenter M:READ (19) to load the overlay. A total of 51 temp memory locations may be used.

3. Normally, M:SEGLD requires 29 temp memory locations. However, 61 are required to output the message !!BEGIN SEG xx. This is an RBM assembly option (i.e., #SEGXX = yes).

4. M:CKREST requires 65 temp memory locations if the checkpoint is performed at the priority level of the calling task and the message !!BKG CKPT is to be typed out. This message can be suppressed if bit 8 of R:SYFG is set, in which case M:CKREST requires 33 temp memory locations.

5. Use of any device that has a nonresident device dependent I/O edit or error recovery routine associated with it requires 51 temp memory locations by M:READ/M:WRITE. These include KP, PT, LP, B7, CR, and CP. However, if one of these devices is not ready, 83 temp memory locations may be required.

Certain Monitor service routines are nonresident overlay routines. The Monitor subroutine Q:ROC controls the loading of the RBM overlay area. The following Monitor service routines are nonresident overlay routines:

M:ASSIGN	M:DATIME	M:OPEN
M:CLOSE	M:DEFINE	M:RSVP
M:COC	M:DOW	M:WAIT
M:CTRL	M:LOAD	

Actually, portions of the above routines are resident. The resident portion of M:CLOSE, for example, is as follows:

```
M:CLOSE     RCPYI     P,T
            B         Q:ROC
            DATA      'id nn'
```

where

id represents the segment identifier of the nonresident overlay section of M:CLOSE.

nn is the temp stack requirement.

Q:ROC will call M:RES to reserve the appropriate amount of temp space, will read in the required segment, and will transfer control to the overlay routine which runs and returns to Q:ROC. Q:ROC will reload the overlay area if appropriate[t] and will then release the temp space and return to the caller by a call to the Monitor service routine M:POP. Particular attention should be given to the maximum temporary stack requirements of these routines.

SERVICE ROUTINES

M:IOEX (General I/O Driver – SYSGEN optional)

M:IOEX provides direct control by background programs, the Monitor, or foreground real-time programs over all I/O

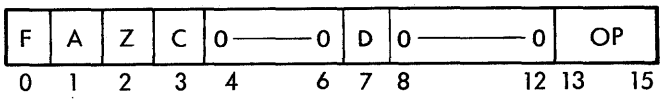
[t]If the overlay area was originally occupied by an active Monitor service routine, the routine must be reloaded. If the requested routine is the one occupying the overlay area, no loading will be required.

operations on the buffered I/O channels for centralization of I/O interrupts. All M:IOEX control functions are exempt from channel time limits. The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:IOEX
```

where adrlst is a pointer to the argument list, which is a set of two, three, four, or five consecutive words in the user's program or in a temporary stack. This argument list appears as follows:

word 0



where

F = 0 if word 1 is an operational label or device unit number.

= 1 if word 1 is a device file number.

A = 1 if AIO Receiver is specified in word 3 (foreground option only).

= 0 if no AIO Receiver is specified.

Z = 1 if AIO Receiver is acknowledged on zero-byte-count interrupt.

= 0 if acknowledged on channel-end only.

C = 1 if a user-written command chaining receiver is specified (foreground option only).

= 0 if no command chaining receiver is specified (default command chaining to be used).

D = 1 indicates that the device has been marked "down" by a DU key-in.

= 0 indicates that the device is not down.

I/O may not be performed on a down device unless bit 7 of the request order word is a one; otherwise, device-unavailable status is returned. Similarly, I/O may not be performed on an "up" device unless bit 7 of the request order word is a zero.

The D bit is intended for the use of RBM diagnostic programs to allow testing of failing devices. User programs should code with the D bit reset (D = 0).

OP is the code for the operation to be performed:

0 for SIO

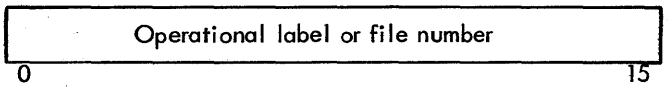
1 for TIO

2 for TDV

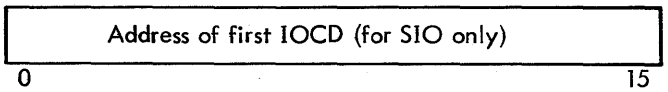
3 for HIO

4 for "check previous data transfer"

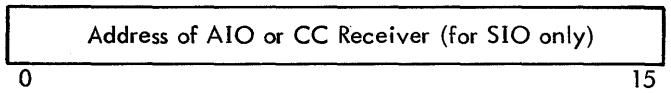
word 1



word 2

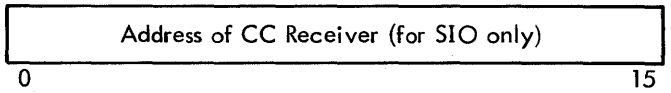


word 3



If bit A = 1 (word 0), then word 3 is a pointer to the AIO receiver. If A = 0 and bit C = 1, then word 3 is a pointer to the command chaining (CC) receiver. If both A = 1 and C = 1, however, an additional word is required as a CC-receiver pointer:

word 4



Return to the user's program is to the location in register L on entry to M:IOEX. Register B is always saved.

The Overflow (OI) and Carry (CI) Indicators, the A register, E register, and (in some cases) X register are used to return status information on the required operation. The complete list of status codes is given in Table 7.

If a device has been declared "down" through the operator's use of a DU key-in, only M:IOEX calls having bit D = 1 (in word 0) are permitted access to the device. This is intended to allow RBM diagnostic programs to test failing devices. Otherwise, (D = 0), a device-unavailable status is returned (rE = -1, rA = 9, rX = 0). Conversely, if a device has not been declared down, IOEX calls with D = 1 are not permitted; a device-unavailable status is returned to the diagnostic program.

Note that no I/O error recovery is attempted. DSBs and OSBs are just as received from the I/O system hardware. These status returns are organized so that a quick and simple test will show the nature of the return. If the user wishes to keep trying to initiate the I/O operation or keep checking for completion, it is possible to loop back to the call to M:IOEX.

Table 7. Return Status from M:IOEX

Operation	Major Status	OI	CI	E Register			A Register		X Register
				0	1 - 7	8 - 15	0 - 7	8 - 15	0 - 15
SIO, TIO, TDV, HIO	Device number not recognized	1	1	0	——		Recognition Code		0
All	Invalid call or oplb	0	0	1	——		4 or 8		0
	Oplb set to zero	0	0	0	——		2		0
	Device unavailable	0	0	1	——		9		0
SIO	SIO cannot be accepted[t]	0	1	0	Current DFN		TIO DSB	Dev. No.	0
	Channel busy[t]	0	0	0	Active DFN		TIO DSB	Dev. No.	-1
	Successful initiation	0	0	0	Current DFN		SIO DSB	Dev. No.	0
TIO	SIO cannot be accepted	0	1	0	Current DFN		TIO DSB	Dev. No.	——
	Other	0	0	0					
TDV	Device abnormal condition	0	1	0	Current DFN		TDV DSB	Dev. No.	——
	Device normal condition	0	0	0					
HIO	Device operating when HIO received	0	1	0	Current DFN		HIO DSB	Dev. No.	——
	Device not operating when HIO received	0	0	0					
I/O check	I/O operation in progress	1	0	0	Current DFN		SIO DSB	Dev. No.	——
	I/O completed unusual end	0	1	0	E Flag (Bit 7)	OSB	AIO DSB	Dev. No.	Byte Count Residue (from even I/O channel register at channel end)
	I/O completed normal end	0	0	0					

[t]Use BXNC to test both conditions simultaneously.

Legend:

DSB = Device Status Byte

Dev. No. = Device number of current device

OSB = Operational Status Byte

If an AIO and/or a CC receiver is specified, it must be a closed subroutine, is executed at the I/O interrupt level, and must return to the I/O Interrupt Task. The same general usage rules govern both: no monitor services may be called, all registers are considered volatile, and processing must be brief so as not to interfere with other on-going I/O. On entry to the CC receiver, register L contains the return location and register X contains the DFN; on exit, if register A is negative, software command chaining will not be performed; if zero or positive, such chaining will take place. (See "M:IOEX Functions", below, for functional descriptions of AIO and CC receiver operation. See also "End Action" in Chapter 5 and "AIO Receivers" in Chapter 6.)

The user can use M:IOEX to read/write on the RAD or any peripheral device that uses standard Xerox peripheral responses. For input/output operations to the RAD, the user must first give a seek order and then the appropriate data-transfer request. The user must also perform his own file management. If multiple tasks use the RAD, they must cooperate in some way so that the seek address is not modified by some higher-level task before the data operation is initiated. Note that a user must always issue a "Check" (op code of 4) after each read or write request.

The following special rules govern the use of M:IOEX for a RAD:

1. A device-file name of the form XXdn must be included in the set of SYSGEN input parameters following the heading DEVICE FILE INFO, where XX indicates that this is a special-purpose device for use with M:IOEX, and dn is the hardware device number of the RAD. The M:IOEX calling sequence must contain the device-file number corresponding to this device-file name, or must contain an operational label that is assigned to the device-file number.

2. The set of SYSGEN input parameters following the heading RAD ALLOCATION must include provisions for reserved tracks that are not to be included in the areas allocated for RBM file management. This can be accomplished by

   a. Assigning the system RAD to a device number other than XXdn. This method requires two RADs, one containing the RBM area assignments, and the other available for use with M:IOEX.

   b. Allocating only part of a RAD for RBM area assignments, leaving the remainder available for use with M:IOEX.

   c. Allocating part of a RAD for M:IOEX use by specifying that a number of tracks be skipped between RBM areas with an allocation parameter of SK = n, where n is the number of tracks.

   d. Any meaningful combination of the above.

M:IOEX FUNCTIONS

TIO, TDV, HIO In these operations, the request is performed immediately and the device status bytes are returned if the device is recognized. The AIO Receiver is ineffective for these operations.

SIO The SIO operation is initiated if there is device recognition and the channel is free (which may not be the same as "device free" or "device controller free" for channels with several devices).

The SIO is issued even if the device is in the manual mode. It is therefore the responsibility of the user's program to test for the manual mode both before and after the SIO request, and to inform the operator by a suitable message.

An HIO can be used to abort an I/O operation. This results in setting the channel end device ready for a new activity. Since status is returned, an I/O check operation is not required.

Protection checks are performed only for background I/O requests. Background is not permitted an AIO Receiver, and a receiver is ignored if requested from the background. Background operations specifying data chaining are not allowed. This is due to the structure of the IOCDs, I/O Data Tables, and the requirements for the absolute protection of foreground programs (see "End Action" in Chapter 5).

The user of M:IOEX must be thoroughly familiar with machine-level I/O operations in general, and in particular with the execution of the SIO instruction as described in the appropriate Xerox computer reference manual. M:IOEX does not modify the user's IOCDs or device-order bytes in any way.

When using foreground data chaining it is very important to set the interrupt flags on all IOCDs, since an unusual end condition in one of the IOCDs without the interrupt flag being set will cause the I/O to terminate without an interrupt, and the channel may then "hang up" waiting for the interrupt because the RBM tables indicate that the channel is still busy.

In addition to hardware-executed data chaining, RBM provides a software convention for command chaining. Its operation and control is analogous to data chaining and involves an extension of the normal, hardware-determined IOCD and I/O-control-table formats. The use of command chaining is fully described in the RBM/System Technical Manual, 90 11 53 (see also the RBM I/O control tables illustrated therein for usage examples). If a command chaining (CC) receiver is specified in the M:IOEX argument list, it is entered at I/O completion time prior to the execution of software command chaining, at the I/O interrupt level. The purpose of the CC receiver is to allow the user to make a real-time decision as to whether or not

a command-chained operation is to be continued. The content of register A on return from the CC receiver overrides standard RBM command chaining control (see below): if the A register is negative, chaining is to be terminated; if zero or positive, continued. (The receiver is entered only if there is another operation to be performed in the user's I/O table.) If no CC receiver is specified, a default RBM receiver is entered and default command-chaining control is exercised: the operational status byte (bits 0-7 of the even channel register) is tested for transmission error, chaining modifier, or unusual end. If any of these conditions is true, command chaining is terminated. Since neither data chaining nor software command chaining operations are permitted for background programs, the specification of a CC receiver from the background is ignored.

The Monitor does not alter the user's data in any way. If an I/O interrupt is received and there is no AIO Receiver specified (and the device is still busy), the I/O interrupt is ignored and the channel remains active.

The user's program must determine whether there was a channel end or an unusual end condition. If the return is for a busy device or channel, the program can loop on this request until the operation is successful.

Since only higher priority tasks can take control from the task issuing the request, the routine issuing the request gains control of the desired device and/or channel as soon as the current operation is complete. The M:IOEX routine inhibits interrupts for a period of less than 100 microseconds during the loading of the I/O channel registers and the setting of the activity status for the device and channel. Thus a higher priority task can always interrupt up to the point when the I/O channels are loaded during the initiation of an I/O request.

I/O CHECK This operation tests for channel end on the previously requested I/O operation by testing certain flags within the RBM I/O tables. The flag is set by the I/O interrupt task when the device interrupt occurs. Thus, no TIOs are required to determine when the operation is complete. Since the TIOs do consume some I/O time (particularly if executed repeatedly in a test loop), the method of checking for I/O completion described herein is desirable. The Monitor saves the operational status byte and the byte count residue from the completion of the I/O operation, even though another device may have used the channel before the end-action check is made by the requesting task.

The following restrictions are pertinent in using M:IOEX:

1. RBM will not necessarily recover automatically from the results of an HIO for most devices. Operator intervention may be necessary.

2. Background programs cannot specify data chaining or command chaining.

3. Background programs must specify an interrupt in all IOCDs.

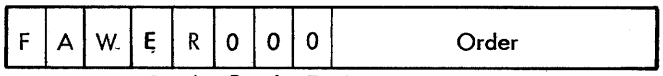
M:READ (General Read Routine)

M:READ provides device-independent input with standard editing and checking. Standard error detection and correction is optional on each call. The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:READ
```

where adrlst is a pointer to the argument list, a set of two to six words in the user's program or in a temporary stack. This argument list appears as:

word 0



where

F = 1 if a device-file number is specified.

= 0 if an operational label or device unit number is specified.

A = 1 if an AIO Receiver address is specified (specifiable by foreground only).

= 0 if no AIO Receiver is specified.

W = 1 if wait for completion is unconditional.

= 0 if wait is for "initiate and return" only; return is immediate if operation cannot be started at once. (The minimum-seek algorithm does not apply to RAD "no wait" operations.)

E = 1 if standard error recovery is to be performed at channel end.

= 0 if no error recovery is to be attempted.

For RAD or disk pack, five attempts for error recovery will be made if E is specified; 10 attempts will be made for magnetic tapes. If I/O without a WAIT is specified, error recovery will not be performed until a "Check" is issued by the user. See RAD and Disk Pack Error Recovery.

R = 1 direct access: a RAD record displacement is specified (granule or logical record number, applicable only to random files). If the file is not random, calling sequence error is returned.

= 0 sequential access: a RAD record displacement is not specified and implies sequential access of random files or sequential files.

If the order is "Check previous output for completion (04)", the 'R' is used as follows:

R = 0 do not retry the operation if operator intervention is required; instead, return "Operator Intervention Required".

= 1 retry the operation, notifying the operator if intervention is required.

D = 1 indicates that the device has been marked "down" by a DU key-in.

= 0 indicates that the device is not down.

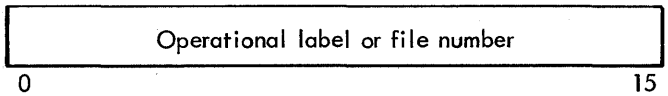
I/O may not be performed on a down device unless bit 7 of the request order word is a one; otherwise, device-unavailable status is returned. Similarly, I/O may not be performed on an "up" device unless biy 7 of the request order word is a zero.

The D bit is intended for the use of RBM diagnostic programs to allow testing of failing devices. User programs should code with the D bit reset (D = 0).

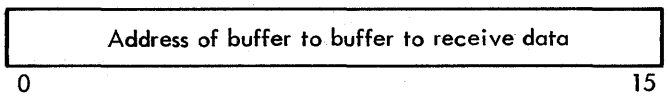
Order is one of the following permissible pseudo input orders:

Order	Operation
X'00'	Return information about this device and file. See Return Registers.
X'02'	Read binary.
X'04'	Check previous input for completion (after a "no wait" initiation).
X'06'	Read automatic.
X'0C'	Read backward (9-track magnetic tape only).
X'10'	Return information on FORTRAN associated files.

word 1

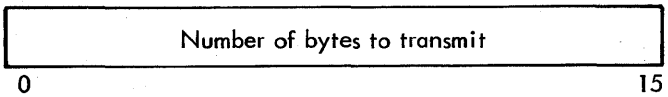


word 2



Buffer must be in background if called by a background program. Also, buffer must not overlap active temporary storage or unavailable memory.

word 3

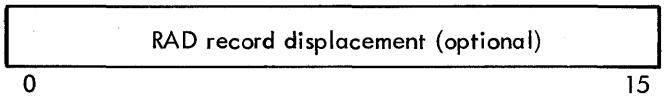


Byte count must be an even number when reading from RAD files and cannot exceed 65,534. For all other devices the byte count may be either even or odd but cannot exceed 8192. If the byte count is even, input data stored in the user's buffer starts in the left-hand byte; if odd, data starts in the right-hand byte.

word 4

If A = 1 (in word 0), this is the address of the closed AIO Receiver subroutine called by the I/O interrupt task at channel end. If A = 0, this is the RAD record displacement (granule or logical record).

word 5



If an AIO address is specified (A = 1 in word 0), word 5 indicates the displacement from the start of the file (starting with a displacement of zero). Transfer starts at the beginning of the indicated file unit. Word 4 is RAD file unit displacement if A = 0.

While blocked and unblocked random files may be accessed directly or sequentially, the usage modes should not be freely mixed. Note that if the R bit is not set for random files, the file is processed sequentially.

RETURN REGISTERS

Return is always to the location specified in the L register. The B register is always saved.

The E, A, and X registers all contain status information on the return, as shown in Table 8. I/O completion codes are listed in Table 9. Return is always immediate if there is a calling sequence error, in which case the E register is negative upon return. For the case where a wait is specified, the I/O is initiated and the M:READ routine loops until the operation is complete. When "initiate and no-wait" is specified, an SIO is issued before the return if the device is recognized, is currently free, can accept an SIO, and is not in the "manual" mode. If any one of these conditions is false, the M:READ routine returns immediately with the appropriate indicators set. If the channel or device is busy, the caller can either loop back to the call to M:READ or switch to another device. The "wait" flag has meaning whether this is an initiate or a check order. Error recovery is attempted if specified before the final return is made.

Table 8. Return Status from M:READ, M:WRITE, M:CTRL

Operation	Major Status	Action	E Reg.	A Reg.	X Reg.
All operations	Operational labels not valid.	Return immediately.	-1	8	tt
	Calling sequence error.	Return immediately.	-1	4	tt
	Operational label is set to zero.	Return immediately.	0	2	t
	RAD or magnetic tape file positioned at EOT.	Return immediately.	0	4	t
	Irrecoverable I/O error.	Return after error recovery attempt, if any.	-1	1	t
	Device has been declared unavailable.	Return immediately.	0	9	t
Initiate	Blocking buffer not available.	Return immediately.	0	10	t
Initiate I/O and no wait	Channel and device are free and in automatic.	Initiate I/O and return. Status in X register only meaningful if A = 1 in the call and the A register is zero upon return. X = -1 if the AIO Receiver will not be acknowledged; otherwise X = 0.	0	0	0 or -1
	Channel and/or device are busy.	Return immediately.	0	-1	t
	Manual intervention is required (manual mode or device not recognized).	Return immediately.	-1	-1	t
	Completion available without I/O being initiated.	Return immediately.	0 or -1	See Table 9	t
Check and no wait	I/O still in progress.	Return immediately.	0	-1	t
	I/O complete.	Return after end-action, if any.	0 or -1	Completion code (Table 9)	Byte count
Initiate and wait	Channel and device are free and automatic.	Initiate I/O and wait for completion.	0 or -1	See Table 9	Byte count
	Channel or device are busy.	Wait and keep trying.			
	Device number is not recognized or is write protected.	Type out the proper message to operator and retry.			
	Device is in manual mode.	Type out EMPTY message to operator and retry.			
Initiate and wait or check and wait	I/O still in progress.	Wait, and keep checking.			
	I/O complete.	Perform any end-action and return.	0 or -1	Completion code	Byte count transmitted

[t] Unspecified.

[tt] Sector size (in bytes) of the device containing the BT area.

Table 9. I/O Completion Codes

E Reg.	A Reg.	Meaning	Comment
0	0	Operation successful.	X register contains the number of data bytes transmitted.
-1	-1	Operator intervention is required. Normally, this error is equivalent to an I/O error.	The operator was appraised during the error recovery procedure that intervention is required.
-1	1	Irrecoverable I/O.	If error recovery was specified, the maximum number of retries have been unsuccessfully attempted.
0	2[t]	Operation not meaningful for this device.	Either an operational label was assigned to file zero or I/O operation is not meaningful for the device.
0	3[t]	End-of-file encountered.	Significant only for magnetic tape and sequential RAD files (except in automatic mode when significant also for cards, paper tape, and keyboard/printer).
0	4[t]	End-of-tape encountered.	Significant only for magnetic tape or sequential and random-access RAD files.
0	5	Incorrect record length.	For read operations, the requested byte count does not equal the device's physical or logical record size. For write operations, the requested byte count is greater than the device's physical or logical record size. For either read or write, the actual byte count transmitted is returned in the X register.
0	6	No I/O pending for this check operation.	Error in I/O buffering. An initial no-wait I/O request either was not issued or was rejected.
0	7[t]	Device is write-protected.	Significant only for writing on magnetic tapes and RAD files.
0	8	Beginning-of-tape encountered.	Significant only for reading backward and for positioning magnetic tapes and sequential RAD files via M:CTRL.
0	9[t]	Device unavailable.	Device was declared "down" through use of DU operator's key-in.
0	10[t]	Blocking buffer unavailable.	Significant only for blocked RAD files.

[t]Status also meaningful under initiate I/O and no wait.

On a check operation, the byte count returned in the X register may not be meaningful if the calling sequence does not specify the same count as the initial read.

If the order code is X'00', the following device status information is returned.

Register	Status Information
A	Device name (EBCDIC):
	RD = RAD/disk file
	KP = keyboard/printer
	PT = paper tape
	CR = card reader
	CP = card punch
	MT = magnetic tape
	LP = line printer
	PL = plotter
	LD = logical device. (E) not meaningful

Register	Status Information
E	TDV device status byte (bits 0-7) and physical device number (bits 8-15).
X	Physical standard record size (bytes) for non-RAD files or granule size for RAD files.

If the code is X'10', the following status information is returned for random or packed files:

Register	Status Information
A	Address of the FORTRAN associated variable (PTR).
E	File units per FORTRAN logical record.
X	File unit in bytes (granule or logical record size).

If a read is attempted to a flawed track in disk pack files, the header of the flawed track is read to determine its alternate. The alternate track is then read as if it were the original.

M:READ FUNCTIONS

M:READ is designed to read one logical record from the specified device regardless of device type and whether the record is EBCDIC or binary. Therefore, M:READ will set up the proper order bytes for the actual device, using the "pseudo order byte" given in the call to M:READ only as a guide. The user may request fewer bytes than are in the record and only this number will be returned in his buffer. However, if more bytes are requested than are in the record, only the bytes in the record will be read. In any case, the actual number of bytes read will be returned in the X register when the completion code is returned, and if this is not the same as the number of bytes requested, an "incorrect length" code will be returned. While it is not always necessary for the user to check all possible return codes, it may be useful to print them out to aid in debugging.

If an attempt to read a record from magnetic tape results in the detection of an irrecoverable transmission error and incorrect length condition, and if fewer than eight bytes were read from tape, it will be skipped and the next record on tape will be read.

Using M:READ, a user can read 80 EBCDIC bytes regardless of whether they come from cards, paper tape, magnetic tape, keyboard/printer, or RAD. M:READ will perform standard editing from paper tape to give a record a format identical to card image output.

By using a "read and no wait" followed later by a "check for input complete" the user can effectively overlap input and compute.

The order code X'00' is used to request information about an unknown device, and may be helpful in determining the optimum blocking sizes to use.

When using M:READ to make requests on a Logical Device (refer to Chapter 5 – I/O Operations for a description of Logical Devices), where Logical Device is used in the sense of a mechanism to facilitate information transfer between tasks independently of real devices, the following observations should be made:

1. Channel timeout does not apply.
2. Read backward is not meaningful.
3. Read Binary and Read Automatic are not differentiated. Only one record, as specified by buffer address and byte count, is transferred per request.

REAL-TIME PRIORITY

All of the I/O routines are reentrant, and any input can be interrupted for a higher-priority task up to the "point of no return" of setting Monitor status flags and loading channel registers. External and internal interrupts are inhibited for up to 100 microseconds of CPU time during the actual SIO sequence. Keeping a high priority task active and looping on an input request to a busy device enables the task to seize control of the channel or device as soon as the current I/O operation completes.

SPECIAL EDITING FOR CARD READER

Read Automatic. Any cards with a "1" and "2" punch in column 1 are automatically read as binary; all other cards are read as EBCDIC or BCD. (For nonstandard binary cards, the user must use "read binary".) It is possible to specify that all cards from a certain file are to be read as BCD and converted by the M:READ routine to EBCDIC before being returned to the user. Since this would apply only to one file, it is possible to read some cards in EBCDIC and some in BCD from the card reader. (BCD card codes are produced by an IBM 026 keypunch, and EBCDIC card codes are produced by an IBM 029 keypunch.) The EBCDIC record size is 80, and the binary record size is 120 bytes.

An incorrect length status is returned if the requested byte count does not exactly match. An "end-of-file" status is returned when an EBCDIC card that begins with !EOD is input into the user's buffer. An "end-of-tape" status is never returned.

Read Binary. An "incorrect length" status is returned if the requested byte count does not equal the maximum number of bytes requested in the calling sequence. The number of bytes requested, up to a maximum of 120, are input in the user's buffer. "End-of-file" and "end-of-tape" status codes are never returned.

SPECIAL EDITING FOR PAPER TAPE OR KEYBOARD/ PRINTER

Read Automatic. All input from paper tape or keyboard/ printer is initiated in a one-byte-at-a-time mode. From paper tape, the read order is always "read ignoring leader". If the first byte is a code of X'1C', X'3C', X'FF', X'9F', X'BF', X'DF', or X'78' (which can only happen with paper tape), the M:READ routine switches to a binary mode and reads up to 119 more bytes (for a total of 120 in the record). The code byte will be the first byte in the user's buffer.

Code bytes are all invalid EBCDIC codes in the sense that they are not printable graphics or control codes. Since they are all supersets of the card reader "1 and 2 punch" rule for column one, the same codes for "read automatic" can be used for the card reader as for paper tape and, in both cases, the code is part of the user's data buffer. If the first byte from the paper tape or keyboard/printer is not one of the binary codes M:READ continues to read one byte at a time until a NEW LINE code is encountered.

When a NEW LINE code is encountered, input transmission is terminated and the line image is filled out with blanks to the requested byte count. The NEW LINE code is not transmitted to the user's buffer. (If a NEW LINE code is the first code in the input line, it is ignored.)

Thus, all EBCDIC records are of variable length, up to the maximum requested or until a NEW LINE is encountered. Further, EOM and cent (¢) have special meanings within the user's data line. An EOM causes the entire line up to the present position (including the EOM byte) to be discarded. A ¢ sign acts like a backspace. For each ¢ sign received, this byte and the byte preceding it are thrown away.

When reading binary records in the automatic mode, 120 bytes are read regardless of the number of bytes requested. For EBCDIC records, the paper tape is read up to and including the NEW LINE code. For either EBCDIC or binary records, not more than the maximum number of bytes requested is transmitted to the user's buffer. The requested byte count must be 80 for EBCDIC records and 120 for binary records. Any other byte counts result in an "incorrect length" status return.

An "end-of-file" status is returned when an EBCDIC record that begins with !EOD is input into the user's buffer.

Read Automatic from Model 4191 or 4193 Keyboard/Printer. A Read Automatic order for a Model 4191 or 4193 keyboard/ printer (Model 530 systems only) causes a prompt character (/) to be printed immediately prior to reading from the keyboard – there is no INPUT light to indicate "read" state. Otherwise, the operation is the same as described above except that a control-H combination causes the entire line to be cancelled (discarded) and control-X is used for the backspace (character-erase) function.

Read Binary from Paper Tape. The Read Binary order for paper tape is "read immediate" unless it is changed to "read ignoring leader" by a PATCH. The physical record size is the number of bytes requested by the user's input. The next record starts immediately following the last byte of the previous record and the requested byte count determines the end-of-record. "Incorrect length" and "end-or-file" status codes are never returned. "End-of-tape" status is not returned, even when the paper tape runs off the reader.

Read Binary from Keyboard/Printer. A read binary order causes the keyboard/printer to read the exact number of bytes specified. RBM performs no editing, and no bytes (including NEWLINE codes) are considered control bytes. "Incorrect length", "end-of-tape", and "end-of-file" status codes are never returned.

SPECIAL EDITING FOR MAGNETIC TAPE

Read Automatic or Binary. Automatic and binary modes are identical on 9-track tape, and M:READ supports only the BCD and packed-binary modes[†] for 7-track tapes. Only the number of bytes requested is transferred to the user's buffer regardless of the physical record. "Incorrect length" status is returned when there are either too few or too many bytes in the input record, and the tape is positioned at the start of the next physical record. "Incorrect length" will not be reported for too many bytes in the input record for 7-track, packed binary tapes.

If the tape is positioned past the end-of-tape marker and error checking is specified, the device is not started and "end-of-tape" status is returned. If error checking is not specified, the device is started, and the status returned at completion is as in Table 10 except that "end-of-tape" status (A=4) is returned if a file mark is sensed. Read backward operations on 9-track tapes are always permitted past end-of-tape.

The Read Backward order produces a buffer with data in an inverted condition. If the tape is at the load point when the Read Backward order is given, no data is transmitted and "BOT" status is returned. Read Backward will be ignored for devices other than 9-track magnetic tape.

SPECIAL EDITING FOR SEQUENTIAL RAD FILES

Read Automatic or Binary. On a RAD, automatic and binary modes are identical. When reading from blocked files, a blocking buffer must be supplied. If the calling program has not specified a blocking buffer, M:READ will call M:OPEN to reserve a buffer from the calling task's buffer pool. If no buffer is available, M:READ exits with a "blocking buffer unavailable" status.

[†]The user should be thoroughly familiar with the BCD and packed-binary mode if 7-track magnetic tape is used. See the Sigma 7-Track Magnetic Tape System/Reference Manual, 90 09 78.

Compressed records are decompressed by M:READ so that only the expanded record, without compression codes, is input into the user's buffer.

A byte count can be requested that is less than, equal to, or greater than the file's logical record size. The number of bytes requested, up to a maximum of the logical record size, is always transferred. If the byte count does not equal the logical record size, "incorrect length" status is returned. In any case, the file is positioned to the next logical record, regardless of the byte count transferred. For compressed files, the requested byte count is compared to the byte count of the expanded record instead of the logical record size. "End-of-file" status is returned when the file is positioned at the logical EOF. "End-of-tape" status is returned when the file is positioned at the logical EOT. This is true whether or not error recovery is specified.

A Read Backward order will be interpreted as a Read order.

SPECIAL EDITING FOR RANDOM-ACCESS RAD FILES

Read Automatic or Binary. Automatic and binary modes are again identical. For unblocked random files, the exact number of bytes requested will be put into the user's buffer and "incorrect length" status will not be returned. One or more granules will be read to satisfy the byte count. RAD space between granules is lost. Unused parts of granules are ignored.

For blocked random files, no more than one record will be transferred. A greater byte count request results in incorrect length. The file will always be positioned at the next record after a successful transfer.

If the Read begins or extends beyond the file's ending boundary, no data is transmitted and "end-of-tape" status is returned. For blocked random files, an end-of-file may also occur. This is true whether error recovery is specified or not.

Note: For all disk files, no transfer will be initiated that crosses a track boundary. Instead, it will be broken into two transfers: one to transfer to the end of the track, and a second to complete the transfer. Therefore, in a "no-wait" operation, a check must be requested to complete the transfer. If an AIO Receiver is specified, it will be entered each time channel end occurs, but it also must be specified in each Check operation call.

M:WRITE (General Write Routine)

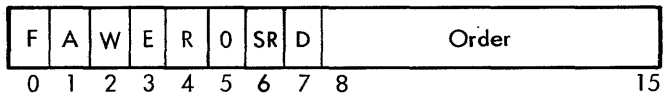
M:WRITE provides device-independent output with standard editing and standard error detection and correction.

The error handling procedure is optional on each call to M:WRITE. The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:WRITE
```

where adrlst is a pointer to the argument list, which is a set of two to six words in the user's program or in a temporary stack. The argument list consists of six words:

word 0



where

F = 1 if a device-file number is specified.

= 0 if an operational label or device unit is specified.

A = 1 if an AIO Receiver address is specified.

= 0 if no AIO Receiver address is specified.

Note: only a foreground operation can specify this.

W = 1 if wait for completion is unconditional.

= 0 if wait is only for "initiate and return"; return is immediate if the operation cannot be started immediately.

E = 1 if standard error recovery is to be performed at channel end for this operation. Five attempts at error recovery will be made for rotating memory devices and ten attempts will be made for magnetic tapes if E is specified. If I/O without a WAIT is specified, error recovery will not be performed until a "Check" is issued by the user.

= 0 if no error recovery is to be attempted.

R = 1 if a RAD record displacement is specified (can only be specified for random-access RAD files).

= 0 if a RAD record displacement is not specified.

If the Order is "Check previous output for completion (04)", the 'R' is used as follows:

R = 0 do not retry the operation if operator intervention is required; instead, return "Operator Intervention Required".

= 1 retry the operation, notifying the operator if intervention is required.

SR = 1 if the user is doing his own blocking to an RBM blocked or unblocked sequential RAD file and an indication of a possible short last record is to be retained in the file directory. If the record being written is actually a short record, a flag will be set in the IOCT for later transfer to the file directory when the file is closed. The actual byte count of the record will be stored into the effective last word of the record. If the record is not a short record, the IOCT flag is cleared; thus this specification is only meaningful for the last record. Upon reading the file, a Read request for the last record (assuming a short record) would result in an incorrect record length status (E = 0, A = 5, X = actual byte count).

= 0 if short record logic is not to be invoked.

The following rules govern the usage of the short record flag:

1. The record must be written from a location that guarantees that the location where the effective last word of the record (as defined by the actual record size) would lie within the domain of the task. This should not be a problem since the record is normally written from the application programs block reserve. Failure to do so from a background program will result in a calling sequence error (E = -1, A = 4). Since the boundaries of a foreground program cannot always be determined, interference with another task can occur.

2. It is assumed that on Read operations the user program is requesting a byte count equal to or greater than the actual record size. A Read request for less than the actual record size would return an incorrect record length for each read and a transfer of the request bytes for each record, including the last record, and may return more data than was actually written into that record, since RBM has no way of determining the written byte count without reading the entire record.

3. The "short record" specification is meaningful only for unblocked sequential and blocked sequential files and is ignored for other devices or files. Only the last record in the file retains the short record indication.

4. The "short record" operation results in a modification to the users buffer if the record is a short record.

D = 1 indicates that the device has been marked "down" by a DU key-in.

= 0 indicates that the device is not down.

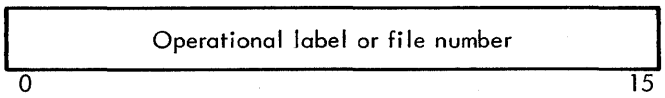
I/O may not be performed on a "down" device unless bit 7 of the request order word is a one; otherwise, device-unavailable status is returned. Similarly, I/O may not be performed on an "up" device unless bit 7 of the request order word is a zero.

The D bit is intended for the use of RBM diagnostic programs to allow testing of failing devices. User programs should code with the D bit reset (D = 0).

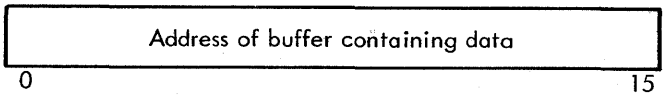
Order is one of the following pseudo order bytes:

Order	Operation
X'00'	Return information about this device.
X'01'	Write binary.
X'03'	Write file mark or !EOD.
X'04'	Check previous output for completion (after a "no wait" initiation).
X'05'	Write EBCDIC.
X'07'	Check write (RAD only).
X'10'	Return information on FORTRAN associated files.

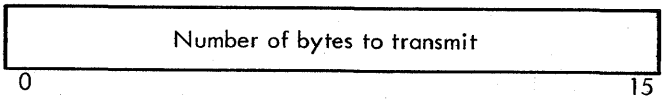
word 1



word 2



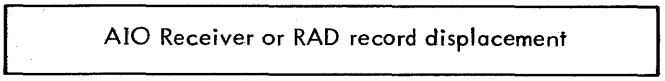
word 3



The byte count must be an even number when writing on RAD files and may not exceed 65,534. It may be either even or odd for all other devices, but cannot exceed 8192 bytes. If an odd byte count is requested, the first byte is written from the right half of the word and the left half is ignored. If an even byte count is requested, the byte is written from the left half of the first word.

Output to the card punch assumes an even byte count. An extra byte at the start of the buffer is sent if the count is odd.

word 4



This is the address of the closed AIO Receiver subroutine called by the I/O interrupt task at the channel end, if A = 1 (word 0). If A = 0, this is the RAD granule displacement (granule or record)

word 5

If an AIO address is specified (A = 1 in word 0), word 5 indicates the displacement from the start of the file (starting with a displacement of zero). Transfer starts at the beginning of the indicated file unit. Word 4 is the RAD file unit displacement if A = 0.

Packed and unblocked random files may be accessed randomly or sequentially. Note that if the R-bit is not set for random files, the file is processed sequentially.

RETURN REGISTERS

The return is to the location in the L register. The B register is always saved.

The status is returned in the E, A, and X registers. Status and method of returning status are the same as for M:READ.

If the code is X'10', the following status information is returned for random or packed files:

Register	Status Information
A	Address of the FORTRAN associated variable (PTR).
E	File units per FORTRAN logical record.
X	File record size in bytes (granule size if random file, logical record size if packed random file).

If a write is attempted to a flawed track in disk pack files, the header of the flawed track is read to determine its alternate. The alternate track is then written as if it were the original.

M:WRITE FUNCTIONS

M:WRITE is designed to write one physical record on the device specified, regardless of the device type. Because of differences in Write orders for the card punch, it is necessary to specify whether the output record is binary or EBCDIC. (For most other devices, the difference is not meaningful.)

Not more than one physical record will be written for a single Write order. For devices like the card punch, if fewer than a standard number of bytes are specified (80 for EBCDIC and 120 for binary), the remainder of the record is padded with blanks (EBCDIC) or zeros (binary). Most of the general comments which apply to M:READ also apply to M:WRITE.

Write End-of-File. Order code X'03' produces the following results:

Device	Result
Line Printer	No effect
Keyboard/Printer	No effect
Card Punch	!EOD card
Paper Tape Punch	!EOD NL
Magnetic Tape	EOF tape mark
RAD	Logical file mark

For devices where the Write End-of-File order has no meaning, a status of "operation not meaningful for this device" will be returned. If a magnetic tape or RAD file is positioned at the end-of-tape, the end-of-file will be output. (This is the only writing allowed past the end-of-tape when error checking is specified.) For RAD files, the end-of-file is set to the current record position within the file as determined by the most recent access through M:READ, M:WRITE, or M:CTRL.

Write EBCDIC to Keyboard/Printer. The first byte is assumed to be a carriage control byte and is never printed. If the byte is a zero or a one, double spacing is used; otherwise, single spacing is used. In any case, this first byte is not sent to the keyboard/printer. Trailing blanks are removed and a NEW LINE code is command chained to the last nonblank byte of the user's buffer. If there are more than 85 printable characters, those beyond 85 are ignored.

Write Binary to Keyboard/Printer. The exact number of bytes specified is written. No format byte is assumed, no editing is performed, and no line format is imposed. It is the user's responsibility to insert NEW LINE codes if more than 85 bytes are output. A maximum of 256 bytes may be output with one operation.

Write EBCDIC to Paper Tape. Trailing blanks are removed and a NEW LINE code is inserted as the last byte (if not already present). The entire record, specified by the byte

count, is edited and output and an "incorrect length" status is never returned.

Write EBCDIC to Line Printer. The first byte per record is always assumed to be a carriage control (format) byte, and is never printed. With any odd byte count (as in all of the I/O), the first byte transmitted is from the right half of the first word, and the left half of the first word is ignored.

The print routine changes the logical format byte (as shown below) to the proper physical format code for the printer. If more than 133 bytes are specified, the remainder beyond 133 bytes is ignored and an "incorrect length" status returned. If fewer than 133 bytes are specified, the right (trailing) portion of the printed image will contain blanks. However, the user's buffer is not modified. The print routine will first data chain on the order byte and format byte in the Monitor area and then on the user's print image.

If it is desired to force single spacing, there may be a word appended to the beginning of the user buffer with a blank in the right half; the byte count is then increased to an odd value, and up to 132 bytes from the original buffer will be printed with the extra "blank" used as the format byte to force single spacing. The format codes (in EBCDIC) are

Format Byte	Effect
blank	No space before printing, single space after printing.
1	Page eject before printing, single space after printing.
0	Single space before printing, single space after printing.
-	No space before printing, no space after printing.

Any other format code will be treated like a blank but will not be printed. These are standard FORTRAN format characters with the exception of the minus sign (-) which is substituted for the standard FORTRAN plus sign (+) to allow overprinting. The user can use M:IOEX (General I/O Driver) to send the standard format code or any other format code for Xerox printers.

Write Binary to Line Printer. Writing binary to the line printer is identical to writing EBCDIC to the line printer except that the first byte from the user buffer is treated as a pseudo VFC and is interpreted by the line printer handler (see Appendix F).

Write EBCDIC to Card Punch. Regardless of the byte count requested, 80 bytes are always output. If fewer than 80 bytes are requested, the punch image is filled out with blanks. The image is moved to a Monitor buffer; the user's buffer is never modified. If more than 80 bytes are requested, only the first 80 are output and the surplus is ignored. In this case, "incorrect length" status is returned. If the file has been declared BCD at system initialization, all EBCDIC output records are converted to BCD before being punched. (The operation is performed in the Monitor's buffer.)

Write Binary to Card Punch. Regardless of the byte count requested 120 bytes are always output. If less than 120 bytes are requested, the punch image is padded with trailing zeros. (The image is moved to a Monitor buffer; the user's buffer is never modified.) If more than 120 bytes are requested, only the first 120 will be output and the remainder ignored. In this case, an "incorrect length" status is returned.

Write EBCDIC or Binary on Magnetic Tape. Variable-length records are possible; no check is made of the data and no editing is performed. The exact byte count (up to the allowable maximum) is always written, however for reliability reasons, it is recommended that byte counts less than twelve or greater than 8190 not be used. For 7-track magnetic tape, the data is recorded in either BCD or packed-binary format, which may cause an "incorrect length" status if the record is not read with the same byte count used to write the record (see the 7-Track Magnetic Tape System Reference Manual, Publication 90 09 78). No "incorrect length" status is ever returned.

If the tape is positioned past the end-of-tape marker and error checking is specified, the data is not transmitted and "end-of-tape" status is returned. If error checking is not specified, the data is transmitted and the "end-of-tape" status is not returned.

If the tape is physically write-protected and an "initiate no-wait" order is requested, the "write-protected" status is returned. If an "initiate and wait" order is requested, the Monitor puts out an alarm and waits for operator action (see the pseudo order bytes under the definition for ORDER under word 0 of the argument list).

Write EBCDIC or Binary on Sequential RAD Files. When writing on blocked files, a blocking buffer must be supplied. If the calling program has not specified a blocking buffer, M:WRITE will call M:OPEN to reserve space in the task's buffer pool. If no buffer is available, M:WRITE exits with a "blocking buffer unavailable" status.

Records to be written on compressed files are edited with compression codes inserted in a Monitor buffer. The data in the user's buffer remains unchanged.

For compressed files only, the logical record size has no meaning and the requested number of bytes is compressed and output. For all other files, a byte count less than, equal to, or greater than the logical record size can be requested and the requested number of bytes, up to the maximum of the logical record size, is always output. If the byte count is greater than the logical record size, an "incorrect length" status is returned. In any case, the file is positioned to the next logical record regardless of the byte count transferred.

An "end-of-tape" status is returned when the file is positioned at the logical EOT (whether error checking is specified or not or if the current operation will cross the logical EOT). Data cannot be output past a logical EOT.

If a Write is attempted on a file that is either logically write-protected or on a RAD track that is physically write-protected, a "write-protected" status is returned and no data is output.

Since the RAD has no read-after-write capability as do magnetic tapes, a separate Check-Write operation is essential to ensure absolute validity of the data output. However, since a separate Check-Write operation requires as much time as the original write operation, and the RAD has a high degree of reliability, the capability should only be used when the data is sensitive or cannot be regenerated. Backspacing operations must be performed before the Check-Write operation, since no repositioning is performed at this time. For compressed or blocked files, no Check-Write is allowed and a status of "operation not meaningful" will be returned.

Write EBCDIC or Binary on Unblocked Random-Access RAD Files. Although a granule size may be specified when a random file is defined, the size does not retrict the maximum number of bytes that may be written. However, each Write operation begins at the start of a granule, and uncompleted granules are filled out with zeros. The exact number of bytes requested is output; never with "incorrect length" status return. If the Write begins or extends beyond the file's ending boundary, no data is transmitted and an "end-of-tape" status is returned, whether or not error recovery is specified.

If a Write is attempted on a file that is either logically write-protected or on a RAD track that is physically write-protected, a write-protected status is returned and no data is output.

Write EBCDIC or Binary on Blocked Random-Access RAD Files. Any access is restricted to the record size regardless of whether the access is random or sequential. Incorrect length and end-of-tape may occur. Write protection considerations are the same as for unblocked random files.

Note: For all disk files, no transfer will be initiated that will cross a track boundary. Instead, it will be broken into two transfers: one to write to the end of the track, and a second to complete the transfer. Therefore, in a "no-wait" operation, a check must be requested to complete the transfer. If an AIO Receiver is specified, it will be entered each time channel end occurs, but it also must be specified in each check operation call which may be different from the AIO Receiver given in the Write call.

When using M:WRITE to make requests on a Logical Device (refer to chapter 5 - I/O Operations for description of Logical Devices), where Logical Device is used in the sense of a mechanism to facilitate information transfer between tasks independently of real devices, the following observations should be made:

1. Channel timeout does not apply.
2. Check Write is not meaningful.
3. Write Binary and Write EBCDIC are not differentiated.

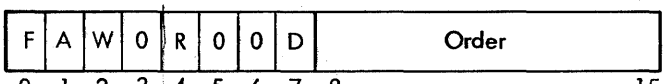
M:CTRL (General Control Routine)

M:CTRL provides device-independent positioning capabilities for magnetic tapes (both 7-track and 9-track) and for RAD files. All M:CTRL control functions are exempt from channel time limits. The calling sequence is

```
LDX     adrlst
RCPYI   P, L
B       M:CTRL
```

where adrlst is the pointer to the argument list which is a set of two or five consecutive words either in the user's program or in a temporary stack. This argument list appears as follows.

word 0



where

F = 1 if a device-file number is specified.
= 0 if an operational label or device unit number is specified.

A = 1 if an AIO Receiver is specified in word 4 (specifiable by foreground only). 'A' is ignored if 'W' = 1.

= 0 if no AIO Receiver is specified.

W = 1 if wait for completion is unconditional.

= 0 if wait is only for "initiate and return", return is immediate if the operation cannot be started immediately.

If the Order is "Check previous output for completion (04)", the 'R' is used as follows:

R = 0 do not retry the operation if Operator Intervention is required; instead, return "Operator Intervention Required".

= 1 retry the operation, notifying the operator if intervention is required.

D = 1 indicates that the device has been marked "down" by a DU key-in.

= 0 indicates that the device is not down.

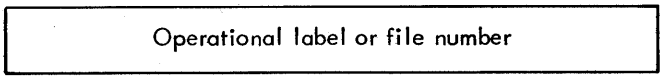
I/O may not be performed on a "down" device unless bit 7 of the request order word is a one; otherwise, device-unavailable status is returned. Similarly, I/O may not be performed on an "up" device unless bit 7 of the request order word is a zero.

The D bit is intended for the use of RBM diagnostic programs to allow testing of failing devices. User programs should code with the D bit reset (D = 0).

ORDER is one of the following pseudo order bytes:

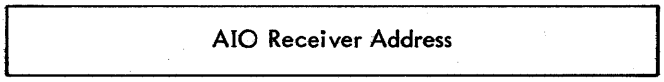
Order	Operation
X'04'	Check previous operation for completion (after a "no wait" initiation)
X'EB'	Space Record Backward
X'EF'	Space Record Forward
X'FB'	Space File Backward
X'FF'	Space File Forward
X'2B'	Rewind Off Line
X'3B'	Rewind On Line

word 1



Words 2 and 3 are currently unused and should be coded as zeroes.

word 4



If A = 1 (in word 0) and 'W' ≠ 1, this is the address of the closed AIO Receiver subroutine entered by the I/O interrupt task when the associated tape motion is complete.

Note: In certain cases, an I/O interrupt will not occur and the AIO Receiver will not be entered. When such a situation exists, M:CTRL will return with the 'X' register set to -1, as for M:READ/M:WRITE functions.

Return is to the location in the L register. The B register is always saved. Status is returned in the E, A, and X registers, as in M:READ. No wait initiate requests must be followed by a check operation. Otherwise, subsequent requests on this file will result in a calling sequence error.

Note: For compressed RAD files, where these operations are not meaningful, an "operation not meaningful" status will be returned.

M:CTRL FUNCTIONS

If the device is a magnetic tape or a RAD file, it is positioned as indicated. The record spacing commands are utilized for physical records and are not meaningful for FORTRAN logical records.

Space Record Backward. The Space Record Backward order positions a magnetic tape to the start of the previous physical record. If the tape is already at load point, the order is ignored and a BOT status is returned. If the previous record was an end-of-file, EOF status is returned.

For compressed RAD files, this order is illegal and a status of "operation not meaningful for this device" will be returned.

For all other RAD files, the file is positioned to the start of the previous logical record. If the file is positioned at the logical BOT, the order is ignored and a BOT status is returned. If the file is positioned immediately beyond the logical EOF, EOF status is returned and the file is repositioned to the point immediately before the logical EOF. If the file is blocked and there is output data in the blocking buffer, it is written on the RAD before the file is repositioned.

Space Record Forward. The Space Record Forward order positions a magnetic tape ot the start of the next physical record. If the record skipped was an end-of-file, EOF status is returned.

For compressed RAD files, this order is illegal and a status of "operation not meaningful for this device" will be returned.

For all other RAD files, the file is positioned to the start of the next logical record. If the record skipped was the logical EOF, an "end-of-file" status is returned. If the file is positioned at the logical EOT, the record is not skipped and an "end-of-tape" status is returned.

Space File Backward. The Space File Backward order positions a magnetic tape to either the start of the previous file mark (and EOF status is returned) or load point (if there is no file mark). If the tape is already at the load point, the order is ignored and BOT status is returned.

For RAD files, the file is positioned to either the start of the logical EOF or to the logical BOT. If the file is positioned immediately beyond or at the logical EOF, it is repositioned to the point immediately before the

logical end-of-file, and EOF status is returned. If the file is positioned before the logical EOF, it is repositioned to the beginning-of-tape and BOT status is returned. If the file is already positioned at the logical beginning-of-tape, the order is ignored and BOT status is returned. If the file is blocked and there is output data in the blocking buffer, it is written on the RAD before the file is repositioned.

Space File Forward. The Space File Forward order positions a magnetic tape to the start of the next file. A status of EOF is returned.

For RAD files, the file is positioned immediately at the logical EOF and "EOF" status is returned. If the file is already positioned beyond the logical EOF or no logical EOF has been written, the order is ignored and an "illegal RAD sequence" status is returned. If the file is blocked and data has been written in the blocking buffer, it will be written out before the file is repositioned.

Rewind On-Line. The Rewind On-Line order rewinds magnetic tape to the load point. If the tape is already at the load point, no error status is returned.

For RAD files, the file is positioned to the logical BOT. If the file is already at the load point, no error status is returned. If the file is blocked and there is output data in the blocking buffer, it is written on the RAD before the order is executed.

Rewind Off-Line. For magnetic tape, the tape is rewound and unloaded. The Rewind Off-Line operation is useful for a "save" tape or for a tape at the end-of-reel when a new tape must be mounted. The user must control and check this condition.

For RAD files, the file is closed by a call to M:CLOSE. If the file is blocked and there is output data in the blocking buffer, the data is written on the RAD before the order is executed. In addition, the file directory is updated on the RAD to reflect the current position of the logical file mark.

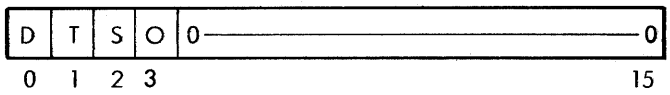
M:DATIME (Calendar Date and Time of Day)

M:DATIME provides the calendar date or time of day, or both, to either foreground or background programs in EBCDIC format. The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:DATIME
```

where adrlst is the pointer to the argument list, which is a set of two consecutive words either in the user's program or in a temporary stack. This argument list appears as follows:

word 0



where

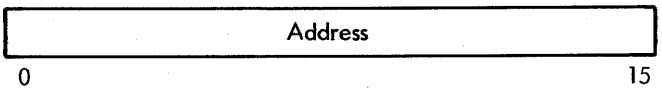
D = 1 if return calendar date is specified.
= 0 if calendar date is not required.

T = 1 if return time of day is specified.
= 0 if time of day is not required.

S = 1 if date and time are supplied by the user (in Address and Address + 1).
= 0 if current date or time of day, or both, are to be used.

O = 1 if date and time are to be unconditionally solicited from the operator.
= 0 if current date or time of day or both are to be used.

word 1

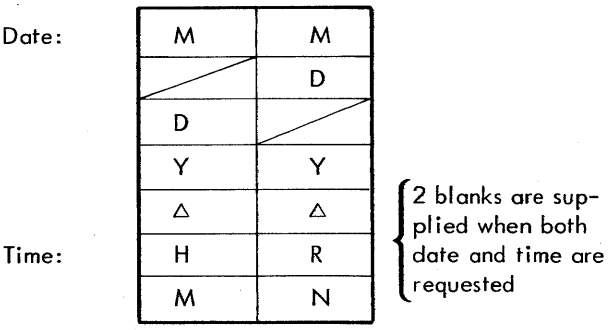


where Address is the location where the date and time of day are stored.

Return is to the location in the L register. The B register is always saved.

M:DATIME FUNCTIONS

K:CLOCK in the communication region is a pointer to the accounting table that contains the date and time. The date and time are set at system initialization and can be reset by the operator through unsolicited key-ins. The date is automatically advanced (if Clock 1 or JOBACCT is indicated) and provisions are included for year changes including leap-year adjustment. Thus, under continuous operation, only adjustments to accommodate daylight savings time changes are required. The date or time of day, or both, are stored in the following format in the area of core specified by word 1 of the argument list:



Note: Time of day is given in military time (0000-2359).

If the date and the time are supplied by the user (S = 1), the times supplied in Address and Address + 1 will be overlaid by the calendar date or time, or both. This option is used by the Job Control Processor !PURGE command.

If O is specified, the date and/or time will be solicited from the operator.

M:TERM (Normal Exit from User Programs)

M:TERM provides an entrance back to the Monitor on a normal termination of a user program. The calling sequence is

```
RCPYI   P,L
B       M:TERM
```

M:TERM FUNCTIONS

For an unload request, M:TERM triggers the RBM Control Task routine S:LOAD for the next load if any other entry is in the queue stack. If no additional requests are present and S:LOAD has checkpointed the background, S:LOAD triggers RBM Control Task S:REST for a restart. Foreground blocking buffers are not closed. A call to M:CLOSE is required before calling M:TERM to guarantee that blocking buffers are correctly merged with RAD files. If the call is from a real-time foreground program, the task is disabled and M:EXIT is called to perform the exit functions. If the calling task occupies nonresident foreground, an unload operation is performed.

On calls from the background the L register must be set to a background address or the background call will be aborted with a protection violation. All I/O is allowed to run down. All files utilizing blocking buffers will have their blocking buffers closed out. If an unconditional postmortem dump was specified, it will be performed at this time. The Control Command Interpreter will then be read into the background and will read the next control command.

M:ABORT (Abort Routine)

When a background program fails for any reason, a call to M:ABORT provides a method of clearing the background program out of core memory and for terminating all active I/O for the background program. The calling sequence is

```
LDA     loc
LDX     code
RCPYI   P,L
B       M:ABORT
```

where code is a word of EBCDIC information and loc is a word of hexadecimal information that is printed on the DO and OC devices to show why the job was aborted.

Return is never to the location in the L register. If the call is from a real-time foreground program, the task is disabled and M:EXIT is called to perform the exit functions. If the calling task occupies the nonresident foreground area, an unload operation will be performed. On calls from the background, the L register must be set to the background or the background call will be aborted with a protection violation. All I/O in progress is allowed to complete and a postmortem dump will be performed at this time if previously requested.

M:SAVE (Interrupt Save Routine)

M:SAVE routine performs the full context switching when a foreground interrupt occurs. It is available only for foreground programs that are connected directly to an interrupt. The calling sequence is

```
RCPYI   P,L
B       M:SAVE
ADRL    tcb
```

where tcb is the address of the Task Control Block for the task.

Return is to the value in the L register + 1. The contents of all registers except A and L are transferred to the TCB.

M:SAVE FUNCTIONS

The contents of A and L must be saved in the proper place in the TCB before the task calls M:SAVE. M:SAVE then saves the original value of X, T, B, and E in the TCB. The interrupting task has its own floating accumulator set into locations 0001-0005 and the previous task's floating accumulator pointers are saved. The M:SAVE routine stores the temporary stack and TCB pointers in locations 0006 and 0007 for this current task and saves the old values in the interrupting task's TCB.

If the flag in the TCB is set for "no temporary storage" M:SAVE saves only the hardware registers and the TCB pointers, and not the full context.

If JOBACCT has been specified, M:SAVE will switch charges to foreground at the first interrupting foreground task.

An additional entry point, M:FSAVE, is available for users of the Store Multiple instruction[†]. This entry point, with an address literal in cell X'C7', assumes that all registers

[†]Store Multiple is a standard feature on Xerox Model 530 and is an optional feature on Xerox Sigma 3 computers.

have been saved, but performs the remainder of the functions of M:SAVE as listed above. The calling sequence is

```
RCPYI   P,L
B       *X'C7'
ADRL    tcb
```

where tcb is the address of the Task Control Block for the task.

M:EXIT (Interrupt Restore Routine)

M:EXIT restores the contents of all registers prior to exit from a foreground task, switches the full context back to the previous task, and performs the actual exit sequence. The calling sequence is

```
RCPYI   P,L
B       M:EXIT
```

Return is to the interrupted task at the address saved in the PSD. All registers are restored to the same value they had at the time of the interruption.

M:EXIT FUNCTIONS

The operations performed by M:EXIT are essentially the reverse of those in M:SAVE. It is necessary to inhibit interrupts for about 11 microseconds for the actual exit sequence, but it is not necessary to call M:EXIT to perform the exit sequence if it can be performed by the user's program.

The TCB contains a flag to indicate whether any temporary storage is used. If the task does not use any Monitor I/O routines or the floating accumulator, no temporary storage is needed. In this case, only the hardware registers are restored. M:EXIT will restore charges to background if JOBACCT has been specified and return is to background.

M:HEXIN (Hexadecimal to Integer Conversion)

The M:HEXIN routine converts a hexadecimal number (represented in EBCDIC) to a binary integer. The calling sequence is

```
LDA     left
RCPY    A,E
LDA     right
RCPYI   P,L
B       M:HEXIN
```

where left and right contain the EBCDIC codes for the hexadecimal number (the left and right part of a possible four-byte field).

Return is to the location in the L register. The result is in the A register, the X register is changed, and the B register is unchanged.

M:HEXIN FUNCTION

Blanks and zeros are treated as hexadecimal zeros. No temporary storage is used and no error checking is performed.

M:INHEX (Integer to Hexadecimal Conversion)

The M:INHEX routine converts a binary integer to a hexadecimal representation in EBCDIC code. The calling sequence is

```
LDA     integer
RCPYI   P,L
B       M:INHEX
```

where integer is the value to be converted.

Return is to the location in the L register. On return, the E register contains the leftmost two bytes, and the A register contains the rightmost two bytes. The X register is changed, but the B register is unchanged.

M:INHEX FUNCTION

Four fields of four-bit hexadecimal codes are converted to four fields of eight-bit EBCDIC equivalents. No temporary storage is used.

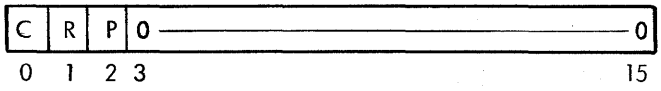
M:CKREST (Checkpoint/Restart Background)

M:CKREST checkpoints the background (i.e., writes it out into a predefined area on the RAD), turns the background space over to the foreground program, and then restarts the background when requested. The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:CKREST
```

where adrlst is a pointer to an argument list, as follows:

word 0



where

C = 1 if request is to "checkpoint" the background.

= 0 if request is to "restart" the background.

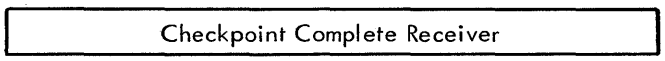
R - 1 if a Checkpoint Complete Receiver is to be informed when the checkpoint is complete. (Valid only if C = 1 and P = 0.)

= 0 if no Checkpoint Complete Receiver is used.

P = 1 if checkpoint is to be performed at the level of the calling task (meaningful only if C = 1).

= 0 if checkpoint is to be performed at the level of the RBM Control Task (meaningful only if C = 1).

word 1



The Checkpoint Complete Receiver should be used like an AIO Receiver. That is, after requesting a checkpoint, the foreground program should release control by a call to M:EXIT and regain control through the specified receiver address when the checkpoint operation is completed. Only a foreground program can checkpoint the background; a background program cannot checkpoint the background area.

Return is always to the location contained in the L register. The B register is always saved. The A register contains the status (1 if operation is impossible; 0 if successful).

M:CKREST FUNCTIONS

Checkpoint. All active I/O for the background is allowed to complete but no error recovery is performed for this I/O until the background is restarted. Peripheral devices dedicated to the background should not be repositioned.

When all I/O has terminated, the entire background space is written out onto a prespecified area of the RAD and the background is set "protected". If the background is truly "empty"[†] when the request is made, the checkpoint is performed immediately, and no RAD is required for the checkpointing procedure. If a Checkpoint Complete Receiver was specified, it will be entered with the L register set to the return address and will be run at the RBM Control Task level.

A checkpoint operation will be automatically performed while loading a nonresident foreground program that extends into the background. When the active nonresident program unloads (see Monitor service routine M:LOAD), the background will be automatically restarted. When the checkpoint operation is completed, the message !!BKG CKPT is output to inform the operator.

[†]This would occur after a !FIN command was encountered or when the Monitor was in an idle state after an abort of an attended job.

Restart. A restart is always performed at the priority level of the RBM Control Task. It is assumed that no peripherals have been repositioned. The core allocation table is restored to the previous value before the checkpoint took place, and the background is then loaded in from the RAD and continues as before.

If no background program was in progress when the checkpoint was called for, the background is set to an unprotected status but no attempt is made to reload a program from the RAD when the foreground terminates.

The message !!BKG RESTART is output to inform the operator that the background has been released by the foreground. See Chapter 6 for more details.

M:LOAD (Absolute Core Image Loader)

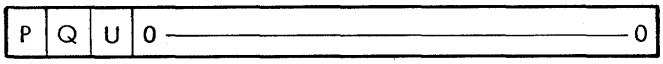
M:LOAD initiates the loading of the root segment of a resident or nonresident foreground program by entering the requested program name into the queue stack. It also initiates the loading of the root segment of a resident or nonresident foreground program or background processor upon request from the Job Control Processor, or from a background program that desires to load and transfer control to another background program. M:LOAD is also used to release (unload) the nonresident foreground space for use by the next program in the queue.

The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:LOAD
```

where adrlst is a pointer to an argument list, as follows:

word 0



where

P = 1 indicates a request to read from the specified device-file number (word 1). The device-file number must currently be assigned to a RAD file. (This option is restricted for use by the Job Control Processor.)

= 0 indicates a request to read the specified program from the user's processor (UP) RAD area. The program name is given in C1-C8.

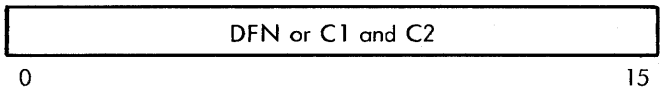
Q = 1 indicates the request is to be queued if it cannot be satisfied now (meaningful only for foreground loads).

= 0 indicates the request is to be ignored if it cannot be satisfied now.

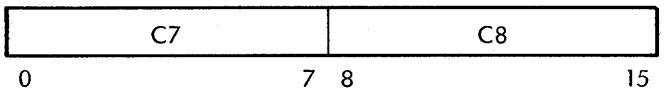
U = 1 indicates an unload operation, in which case P and Q are not meaningful.

= 0 indicates a load operation.

word 1



word n



where

DFN is the device-file number.

C1-C8 is the program name (must be 8 characters, including trailing blanks; program must reside in the UP area).

For foreground loads, return is always to the location in the L register. The contents of the B register are always saved and the A register contains status codes, as follows:

A Register	Meaning
0	Operation is successful.
1	Request cannot be honored at this time (this could occur if Q = 0 and a nonresident foreground area was already committed; or if Q = 1 and the queue stack was full).

M:LOAD FUNCTION

After saving the nonresident program name or device-file number request, M:LOAD triggers the RBM Control Subtask S:LOAD and then exits to the location in the L register.

The actual loading of the program is accomplished at the priority level of the RBM Control Task. S:LOAD will ensure that sufficient blocking buffers are available for those operational labels contained in the header record of the processor. If the request was for a nonresident foreground program that extends into area reserved for the background, S:LOAD automatically causes the background to be checkpointed.

If the request is from a background program, a load and transfer control operation is assumed. Blocking buffers from the current blocking buffer pool will be closed. All operational labels except PI will retain their current assignments. The contents of COMMON and CCBUF will be retained. The X register, upon transfer to the new background program, will point to CCBUF; all other registers are volatile. Operational label PI will be assigned to the new task for SEGLOAD operations.

It is essential that each nonresident program executed in the nonresident foreground area terminate itself by a call to M:TERM to unload, disable itself, and then exit via the normal interrupt exit routine (M:EXIT). This will release the nonresident foreground area for subsequent loads. An unload request is an implied call to M:TERM and is an alternate way of terminating a nonresident foreground task. M:LOAD will return an error if the calling task is not the nonresident foreground task.

For an unload request, M:TERM triggers the RBM Control Task routine S:LOAD for the next load if any other entry is in the queue stack. If no additional requests are present and S:LOAD has checkpointed the background, S:LOAD triggers RBM Control Task S:REST for a restart.

Note that M:LOAD inhibits interrupts for a short period while manipulating the queue stack (less than 100 μsec if no more than eight entries are waiting in the queue).

M:OPEN (RAD File Open)

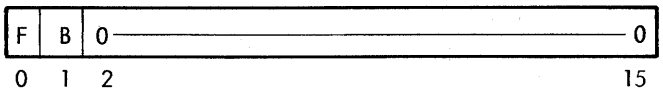
M:OPEN reserves a blocking buffer from a buffer pool or a specified location, for a blocked, compressed, or packed RAD file to which an operational label or device unit number had previously been assigned.

The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:OPEN
```

where adrlst is a pointer to the three-word argument list shown below.

word 0



where

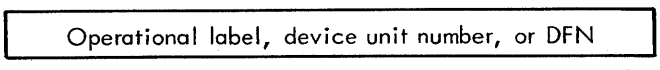
F = 1 if a device-file number (DFN) is specified (internal Monitor calls only).

= 0 if an operational label or device unit number is specified.

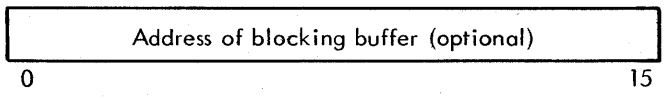
B = 1 if a blocking buffer location is included in this call.

= 0 if no blocking buffer location is included, in which case M:OPEN attempts to find space in the task's buffer pool (see "Blocking Buffers", Chapter 5).

word 1



word 2



Return is to the location in the L register. The B register is restored. The following status information is contained in the A register on return.

A Register	Meaning
0	Operation successful.
1	Blocking buffer already defined.
2	No space available in buffer pool.
3	Illegal operational label or operational label unassigned.
4	Not RAD file, or not a blocked RAD file.
5	Blocking buffer outside of background for a file assigned to the background.
6	Illegal DFN.

M:OPEN FUNCTION

The address of the blocking buffer (either the one specified or one located from the task's buffer pool established by an ABS or $BLOCK command) is stored in the RAD I/O Control Table. If no open request has been performed for a blocked, compressed, or packed file by the user's program, M:READ, M:WRITE, or M:CTRL will call M:OPEN to allocate a buffer from the blocking buffer pool on the first data transfer operation.

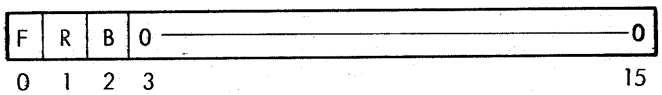
M:CLOSE (RAD File Release)

M:CLOSE releases a RAD file (including the blocking buffer if any) or releases the blocking buffer for a blocked file, but retains the file assignment. In either case, partially filled blocking buffers are written onto the RAD. The calling sequence is

```
LDX     adrlst
RCPY1   P,L
B       M:CLOSE
```

where adrlst is a pointer to the argument list, as follows:

word 0



where

F = 1 if a device-file number is specified.

= 0 if an operational label or device unit number is specified.

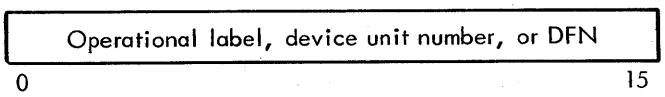
R = 1 if the device-file number is to be released.

= 0 if the device-file number and operational label remain assigned but the blocking buffer is to be released (the file is not to be repositioned).

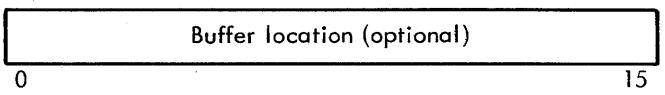
B = 1 if a buffer is specified.

= 0 if no buffer is specified.

word 1



word 2



Return is always to the location in the L register. The B register is always restored to its former value. The A register contains one of the following completion status codes.

A Register	Meaning
0	Successful.
1	Illegal DFN.
2	The operational label is not assigned to a RAD file.
3	Illegal operational label.
4	I/O error writing blocking buffer or EOF onto RAD.
5	No buffer available to complete the close operation.

M:CLOSE FUNCTIONS

If the file is blocked and data has been written on it, the contents of the blocking buffer are written onto the RAD.

If the blocking buffer was allocated from the task's buffer pool, the buffer is released. The EOF is written on the RAD.

If R = 1, F = 0, and the operational label has a permanent assignment, the label is set to that value. If the label has no permanent assignment, the label is deleted from the table of operational labels.

If an EOF has been written on the file it must also be written onto the RAD. To accomplish this, M:CLOSE

requires a buffer into which the file directory is read. If no buffer is specified, M:CLOSE attempts to allocate a buffer from the task's buffer pool (or will use the one already opened for this file if it is blocked). If no buffer is available and an EOF is to be written, the file is not closed and an error completion code is returned.

If a file to be released happens to be last allocated in the Background Temp area (BT), its space will be recovered. Therefore, if BT files are closed in the reverse order from which they are allocated, Background Temp space may be recovered.

M:DKEYS (Read Data Keys Routine)

M:DKEYS provides a means for background programs to read the data keys on the processor Control Panel. The calling sequence is

```
RCPYI   P,L
B       M:DKEYS
```

Return is to the location in the L register. The contents of the B register are always saved. The contents of the data keys are in the A register on return.

M:WAIT (Simulated Wait Instruction)

M:WAIT provides a means for background programs to execute a Wait instruction from nonprotected memory. The calling sequence is

```
RCPYI   P,L
B       M:WAIT
```

The return is to the location in the L register. The B register is always saved. The return does not take place until the operator performs an unsolicited S key-in.

The Monitor types out the message

```
!!BEGIN WAIT
```

and goes into a wait loop.

Only a background program may use M:WAIT; a call from a foreground program results in a no-operation.

M:SEGLD (Load Overlay Segments)

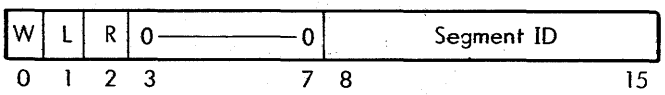
M:SEGLD loads and/or executes an overlay segment, for either the foreground or background, from a file previously prepared and saved on the RAD by the Overlay Loader or the Absolute Loader.

The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:SEGLD
```

where adrlst is a pointer to the argument list.

word 0



where

W = 1 if an unconditional wait for completion is specified.

= 0 if loading is to be initiated only; control will be returned to the calling program.

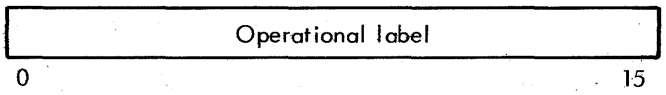
L = 1 control is to be transferred to the transfer address (if one exists) of the segment just loaded, in which case the L register is not meaningful when the transfer occurs (valid only if W = 1).

= 0 control is to be returned to the calling program.

R = 1 there is a "loading complete" receiver (meaningful only if W = 0).

= 0 no "loading complete" receiver.

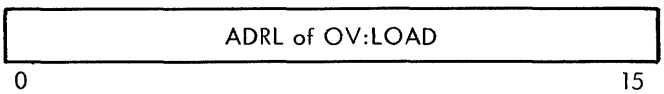
word 1



The operational label is used to control the loading of the segment. The file must previously have been defined as a RAD file and set to the proper overlay program on the RAD. Background programs should use operational label PI.

For the benefit of segmented foreground programs, the initialize code (entered by M:LOAD) can assign an internal operational label to the foreground ML operational label. This internal operational label may then subsequently be used in calls to M:SEGLD. The foreground program may not use the ML operational label in calls to M:SEGLD.

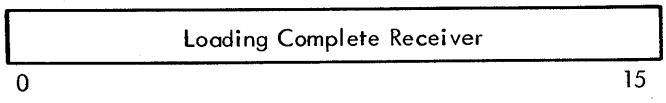
word 2



The symbol OV:LOAD must be declared as an external reference and is set by the Overlay Loader to the value of the Overlay Loader Control Table address in core.

If the program is assembled in absolute form, the Absolute Loader will create the OV:LOAD table at the end of the root. Therefore the last item in the root would normally be an OV:LOAD EQU $.

word 3



The Loading Complete Receiver is permissible only for foreground programs and should be used in the same way as an AIO Receiver. That is, after loading is initiated the foreground program should release control by a call to M:EXIT and regain control through the specified receiver address when the overlay operation is completed.

On all calls specifying an "initiate only", a check operation must be performed on the operational label designated to determine the status of the load and to release the associated device-file number for subsequent use.

On entry, return is to the location in the L register if the L parameter in word 0 of the calling sequence is "0"; otherwise, control is returned to the newly loaded segment. The B register is always saved. On the return, the A register contains status showing the completion code, as follows:

A Register	Meaning
0	Operation complete and successful.
-1	Irrecoverable I/O error (if W = 1), or device containing overlay is busy (if W = 0).
2	Invalid call.

M:SEGLD FUNCTIONS

A core table of 5n + 1 words is maintained at the end of the user's root segment that defines the actual RAD addresses for the overlay segments. (OV:LOAD points to this table; n is the number of segments in the program.) The segments may be loaded in any order because of the random-access capability of the RAD. Using the Loading Complete Receiver and associated procedures can achieve greater efficiency in foreground loading.

M:DEFINE (RAD File Definition)

M:DEFINE allocates a portion of the background temporary file area on the RAD for temporary use by the designated operational label or device unit number. This call is applicable to foreground operations only if the oplabel or fdun has been previously assigned to a permanent RAD file. The calling sequence is

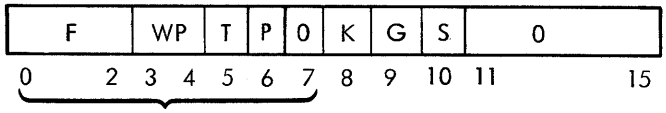
```
LDA     favaa (FORTRAN programs only)
LDX     adrlst
RCPYI   P,L
B       M:DEFINE
```

where

favaa signifies the FORTRAN associated variable absolute address. It is meaningful only if K = 1.

adrlst is a pointer to a four-word argument list.

word 0



where

F specifies the file format as follows:

000	Blocked
001	Compressed
010	Unblocked
100	Random, blocked
110	Random, unblocked

WP = 11 if RBM write protection is specified.

= 10 if foreground write protection is specified.

= 01 if background write protection is specified.

= 00 if write protection is not desired.

T = 1 if the last temporary file allocated is to be truncated so that it will be only as long as its EOF. If no EOF has been written on this file, it will be truncated so that it will be only one record long. Space recovered in this fashion can be reused in the current M:DEFINE call.

= 0 if no truncate is to occur.

P = 1 if word 2 contains a number between 0 and 101 that specifies the percentage of remaining background temporary area to be allocated for this file.

= 0 if word 2 is the number of logical records to allocate.

K = 1 if the A register contains the FAVAA.

= 0 if FAVAA is not specified.

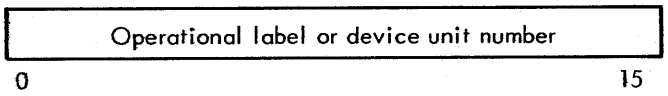
G = 1 if a granule size for random files is specified; otherwise, the granule size is determined by

the sector size of the reference device (meaningful only if F = 110).

S = 1 indicates the file (if packed format) may use the sharable blocking buffer if provided by the Task Control Block (see "Blocking Buffers", Chapter 5).

= 0 indicates sharing is not desired.

word 1

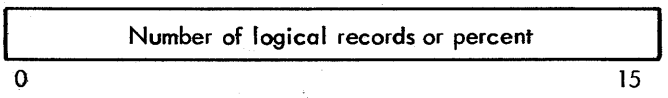


where

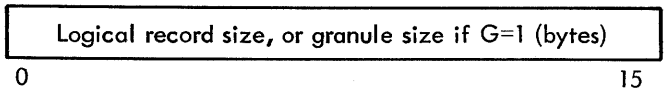
operational labels are EBCDIC

device unit numbers are binary

word 2



word 3



The number of logical records in the file and the logical record size are used to calculate the actual temp space required. For compressed EBCDIC files, the logical record size must be less than 2047 bytes. For compressed EBCDIC files, n card images can normally be accommodated by n/3 80-byte records. Thus, 12,000 card images would require 4000 80-byte records (about 83 tracks on a 360-byte per sector RAD). For blocked, uncompressed files, the total area in sectors equals the number of records requested, divided by the number of logical records per sector. Thus, 120-byte binary card images can be placed three per sector on a 360-byte-per-sector RAD. A 300-card deck would therefore require 100 RAD sectors (seven tracks). If G = 1 and F = 110, the file size is computed using the granule size in word 3.

If this is a random file and G = 0, then the logical record size is actually the FORTRAN random I/O logical record size and the granule size is equal to either the physical sector size for temporary files, or to the granule size defined at file ADD time for permanent files.

For unblocked records, the total area in sectors equals the number of records requested multiplied by the number of sectors required for each record.

Return is to the location in the L register. The B register is restored. The A register contains status information on the return, as follows:

A Register	Meaning
0	Operation successful. E register contains number of records in file; X register contains record size in bytes.
1	Calling sequence error. Logical record size is not an even number or 0 records requested.
2	Operational label invalid (foreground) or no spare entry in operational label table.
3	No more device-file numbers for the RAD.
4	RAD overflow (files too large).
5	K = 1 and attempted to define previously defined file with a different FAVAA. E register contains number of records in file; X register contains record size in bytes.

M:DEFINE FUNCTIONS

For the specified temporary file, the appropriate size is allocated from the pool of temporary file space if such space is available. An unused device-file number is then initialized with the boundary points of this RAD file. All subsequent references to this file (until closed by a call to M:TERM, M:ABORT, or M:CLOSE) will refer to the allocated area. The file is set to the "rewound" condition, if it is a sequential file.

If the operational label is already assigned, no error status is returned if it is assigned to a background RAD file. If K = 1, the address of the FORTRAN Associated Variable from the call must be the same as the one for the file.

Note: M:DEFINE uses locations 1-3 (of the calling program's floating accumulator) for temporary storage.

M:ASSIGN (Assign Operational Label)

M:ASSIGN performs equivalence between an operational label or FORTRAN device unit number, and

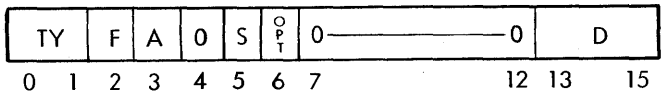
1. A RAD area.
2. A file name within a RAD area.
3. A device-file number.
4. Another operational label or device unit number.

The calling sequence is

```
LDX     adrlst
RCPYI   P,L
B       M:ASSIGN
```

where adrlst is a pointer to an argument list of two to eight words, as follows:

word 0



where

TY = 00 if the label or device unit number is to be assigned to another label or device unit number.

= 01 if the label or device unit number is to be assigned to a device-file number.

= 10 if the label or device unit number is to be assigned to a RAD area.

= 11 if the label or device unit number is to be assigned to a file within a RAD area.

F = 0 if the label is a background operational label.

= 1 if the label is a foreground operational label.

A = 1 if the two-letter area mnemonic is contained in word 3; otherwise, D will specify the area. If A is set, D will be ignored. A must always be set for areas other than SP, SD, SL, UP, UD, UL, BT, and CP.

S = 1 indicates the file (if packed format) may use the sharable blocking buffer if provided by the Task Control Block (see "Blocking Buffers" Chapter 5).

= 0 indicates sharing not desired.

opt = 1 indicates that device specific options are present in words 3-N (meaningful only if TY = 00 or 01).

= 0 indicates that device specific options are not present.

If TY = 00 or 01 and opt = 1, then D = number of device specific options that are present in word 3 to word N. Device options are one- to four-character EBCDIC fields, two words per specification, which are left-justified and blank filled. D must be in the range $0 \leq D \leq 7$.

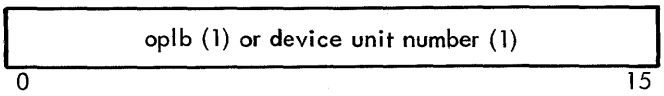
If TY = 10 or 11 then D has the meaning given below.

D = directory to be used:

000	Checkpoint area (area only)
001	System Processor area
010	System Library area
011	System Data area
100	Background Temp area (area only)
101	User Processor area
110	User Library area
111	User Data area (UD only)

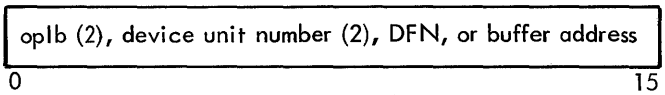
No named files may exist in either the Checkpoint or Background Temp areas.

word 1



where oplb (1) is the operational label or device unit to be assigned.

word 2



where

oplb (2) if present, indicates that oplb (1) will be assigned to the device-file number that oplb (2) is currently assigned to.

DFN if present, is the device-file number that oplb (1) will be assigned to.

buffer address is the first word address of a buffer (equal to one blocking buffer in length) that will be used by M:ASSIGN as temporary storage for the appropriate RAD area dictionary. This is meaningful only for TY = 11.

If TY = 00 or 01 and opt = 1

words 3 and 4

Option 1, C1	Option 1, C2
Option 1, C3	Option 1, C4

(if D = 1 or 2)

words 5 and 6

Option 2, C1	Option 2, C2
Option 2, C3	Option 2, C4

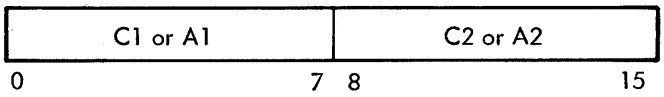
(if D = 2)

Device specific options are represented as a one- to four-character EBCDIC field, left justified and blank filled. Note that the device specific options are meaningful only for certain devices. Use of an unrecognized option for a device results in an error return of "INVALID OPTION".

If TY = 10 or 11, the following options are recognized for Model 3325/35 tape drives:

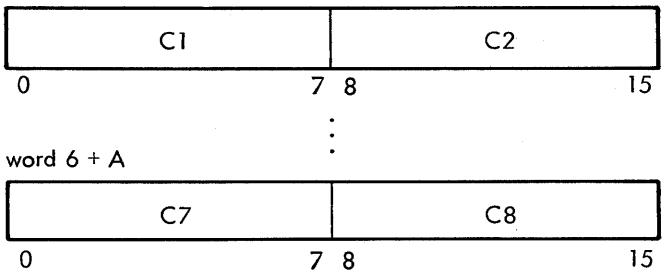
800	For 800BPI, NRZI recoding
1600	For 1600 BPI, phase encoded recording
ASCI	For ASCII code conversion
EBCD	For EBCDIC (ASCII code conversion 'off')

word 3



If A (of first word of argument list) = 1, word 3 contains the two-letter area mnemonic, A1 and A2; otherwise, word 3 contains the first two characters of the file name, as continued below:

word 3 + A



C1-C8, if present, is the name of the file to which oplb (1) is to be assigned. That is, this file on the RAD is to be linked to an unassigned RAD device-file number to which oplb (1) is, in turn, assigned. This is meaningful only for TY = 11.

Return is to the location in the L register. The B register is restored. The A register contains status information on the return as follows:

A register	Meaning
0	Successful operation.
1	Mixed oplbs or device-file numbers (foreground to background or vice versa) or protection violation on buffer address.
2	Invalid oplb or DFN.
3	No spare entries in oplb or DFN tables.
4	File name not found in designated directory.
5	RAD area not allocated.
6	Illegitimate RAD file format.

When the A register = 0, the X register will contain the physical record size (or sector size) for this device and the E register will contain the newly allocated DFN.

M:ASSIGN FUNCTIONS

M:ASSIGN may be called to make any of four types of assignments, according to the setting of TY, as follows:

TY = 00 oplb (1) is assigned to the DFN to which oplb (2) is currently assigned. Oplb (2) must be the same mode (foreground or background) as oplb (1) (error return A = 1). A background program cannot assign foreground oplbs (error return A = 1).

= 01 oplb (1) is assigned to the specified DFN. DFN must be legal, must not be a RAD DFN, and may not be foreground if oplb (1) is background.

= 10 oplbl (1) is assigned to a currently unused RAD DFN which, in turn, is linked via the RBM Master Dictionary to a current RAD area. The area may then be used exactly like a RAD file with the following characteristics:

Format:	random
Logical record size:	sector size in bytes
Write protection:	area write-protect code
BOT:	BOT of area
EOF:	none
EOT:	EOT of area

= 11 oplb (1) is assigned to a currently unused RAD DFN, which in turn is linked via the RAD dictionaries to an individual file within an area (e.g., XSYMBOL). The RAD area must currently be accessible (error return A = 5). The buffer address must be in the background if the calling program is a background program.

If there are no errors, the assign will take place regardless of the prior status of oplb (1). For TY = 10 and 11, RAD files are rewound (file pointer is set to BOT). For TY = 00 and 01, the file position is unchanged.

M:RES (Temporary Storage Allocation)

M:RES allocates storage from a task's temporary stack by addressing the B register to the first available memory location of that stack. If the temp storage is to come from the task's associated temp stack (temp stack pointers in TCB words 3 (start), 4 (end) and 13 (current pointer, K:DYN) it is called dynamic temp. When dynamic temp is requested, M:RES saves the current B register, addresses B to the value in K:DYN (from the TCB) and sets K:DYN to K:DYN+n (where n is defined below).

Monitor service routines use only dynamic temp (as shown in Table 7). This allows them to be reentrant (i.e., used concurrently by different tasks, each with its own unique TCB). The calling sequence to allocate dynamic temp is

```
RCPYI   P,T
B       *$+3
DATA    n
DATA    0
ADRL    M:RES
```

where

T must point to background memory if M:RES is being called by a background program. (Otherwise, a PV abort will occur).

n is the number of memory locations to reserve.

A TS abort will occur if insufficient space is available. This abort can only occur for dynamic temp allocation.

The calling sequence for nondynamic temp allocation is

```
        RCYPI   P,T
        B       *$+3
        DATA    n
NONDYN  DATA    temp    Pointer to nondynamic
                        temp
        ADRL    M:RES
                :
temp    RES     n
```

Nondynamic temp is used traditionally by Basic FORTRAN IV library routines which are not in the Public Library. That is, Basic FORTRAN IV library routines loaded with a specific task, for use only by that task. If one of these routines is to be accessed as a Public Library routine, the OLOAD processor will set NONDYN to zero as it adds the routine to the Public Library and will remove the trailing temp reserve. This trailing TEMP RES n must not be followed by data or instructions.

M:RES FUNCTIONS

The former B register will be saved in location 1 relative to the new B register. Location 0 relative to the new B register will contain 0 if nondynamic temp was specified. Otherwise, location 0 will not be zero and M:RES adds the number of locations requested to K:DYN (i.e., increments the temp stack pointer) after addressing B to the former value of K:DYN. Obviously, locations 0 and 1 relative to the B register must not be changed. Location 2 relative to B is eventually used as the return for M:POP and is initially set by M:RES to point to M:ABORT. M:RES returns to the location in the T register +3 if the reserve was successful; otherwise, M:RES will call M:ABORT with the code 'TS'.

On return from M:RES, the calling program can set its own return through M:POP as follows

```
LDA     = return
STA     2,,1
```

The L and X registers are unchanged on return from M:RES.

M:POP (Temporary Storage Release)

A call to M:POP is made to release the current temporary storage stack (pointed to by the current value in the B register), restore the previous value to B, and return to the location specified in temp + 2.

If the temporary storage was allocated by M:RES, the call must set up a return in temp + 2. The calling sequence is

```
LDA     = return
STA     2,,1
RCPYI   P,L
B       M:POP
```

where return is the location to which return will be made after the stack is released.

Register L must always be set, even for foreground tasks.

Return is to the address specified in location 2, relative to the beginning of the stack being released. The location in the L register and the return address must be in the background area if return is to a background program. On return, B contains its previous value before the RES-POP sequence. Assume return is made to location R; L is set to the value R + 1.

M:POP FUNCTIONS

M:POP performs the opposite functions of M:RES. If location 0 relative to the B register is zero, M:POP does not manipulate the dynamic temp stack pointer, K:DYN. Otherwise, the current value of the B register is stored in K:DYN. Location 1 relative to the B register is then moved to the B register (after 2,,1 is set aside as the return).

M:OPFILE (Convert Operational Label to Device-File Number)

M:OPFILE determines the file to which a foreground or background operational label is assigned. The calling sequence is

```
LDA     type
LDX     adrlst
RCPYI   P,L
B       M:OPFILE
```

where

type is the mode of the operational label; negative for foreground, positive for background.

adrlst is a pointer to the operational label.

Return is to the locafion in the L register. The B register is saved and restored. The status is contained in the E register as follows:

E = negative if label is not found

= positive if label is found

If E is positive, the following information is provided.

Register	Contents
X	Device-file number
E	IOCT entry address[†]
A	Operational label table entry[†]

Note: This routine is used primarily by RBM and certain processors. It will seldom be needed by user programs.

[†]See the chapter on SYSGEN for a discussion of the I/O Control Table and the Operational Label Table.

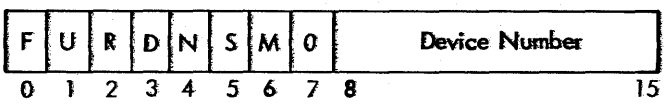
M:RSVP (Reserve or Release Peripherals)

M:RSVP reserves a peripheral device for foreground use only, until the foreground voluntarily releases the device; or until an operator keyin releases the device

```
LDX     adrlst
RCPYI   P,L
B       M:RSVP
```

where adrlst is the pointer to the argument list, which consists of one to four consecutive words either in the user's program or in a temporary stack. This argument list appears as follows:

word 0



where

F = 1 if request is "reserve for foreground".

= 0 if request is "release to background".

U = 1 if request is for an unconditional reserve, where operator intervention is not required.

= 0 if request is for a conditional reserve, where operator intervention is required.

R = 1 if a receiver is to be entered when the conditional reserve is completed (only meaningful if U = 0 or if S = 1).

= 0 if no such receiver is to be used.

D = 0 if a device type is not specified.

= 1 if a device type is specified (used to distinguish KP40 from PT40).

N = 0 if request specifies a device number in bits 8-15 of word 0.

= 1 if request specifies an operational label (contained in word 1 + R + D) which is to be used to determine the actual device number for a reserve operation. The device number upon a successful reserve will be returned in the E-register. The device number must be used for a release operation since an unsolicited 'FL' keyin may have reassigned the operational label.

S = 0 if nonexclusive foreground use of a background device is requested. It is the responsibility

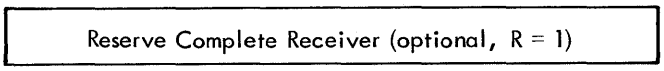
of the user to resolve contention between competing foreground tasks.

= 1 if exclusive use of the device by the requesting task is desired. Since RBM knows the "owner" of the device, an abort or termination of that task will cause an automatic release of the device. Once a task has been granted exclusive use of that device, other requestors receive a "device already reserved" (A = -1) status if R or S = 0, or a return of "request is queued for that device" (A = 3) if both R and S = 1.

M = 0 normal RSVP messages on OC.

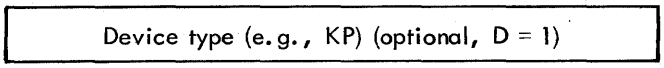
= 1 suppress RSVP messages on OC.

word 1



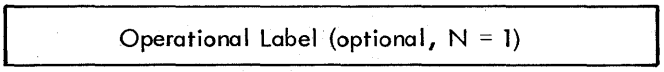
A Reserve Complete Receiver should be used like a AIO Receiver; namely, after the request has been acknowledged, the foreground program should release control by a call to M:EXIT and should regain control when the reserve has been effected through the specified receiver address. This receiver is entered at the priority level of the RBM Control Task and should return to the location contained in the L register.

word 1 + R



If D = 1, word 1 + R contains a device type specification used to differentiate a specific unit of a multiple unit device (e.g., KP40 vs. PT40).

word 1 + R + D



If N = 1, word 1 + R + D contains an operational label to be used for the reserve operation. The actual device number involved will be returned in the E-register.

Return from M:RSVP is always to the location contained in the L register. The A register contains status as follows:

A = 0 if the request is acknowledged. If F = 1 and U = 1 (i.e., unconditional reserve), the device is reserved for foreground use. If F = 0 (i.e., release), the device has been released for background use.

= 1 if the request is acknowledged but operator intervention is required. The Reserve Complete Receiver is entered when the operator effects the reserve. This is the normal response to a conditional request to reserve a peripheral device (F = 1, U = 0, R = 1).

= 2 if the device is not associated with a background file. Not applicable if request was for "exclusive" use.

= 3 request is queued. The RXR (receiver) is entered when the device becomes available (R = 1 and S = 1).

= 4 if the operational label is not defined (Reserve).

= 5 if the operational label is assigned to zero (Reserve).

= 6 RXR (receiver) not specified (F = 1, U = 0).

= 7 if the operational label is assigned to a rotating memory device.

= 8 operational label may not be specified for "release".

= 9 device has been previously reserved as shared.

= -1 if the request cannot be satisfied because the RSVP table is full or if RSVPTABL, 0 was specified at SYSGEN.

X register is significant as follows:

X = 0 if normal condition (i.e., $0 \leq A \leq 3$).

= -1 if abnormal condition (i.e., $A < 0$ or $A > 3$). Thus, a BIX may be used to detect any error.

The E register is meaningful only when request was to reserve via operational label. In this case, upon a successful reserve request return (i.e., X = 0 and $0 \leq A \leq 3$), the E register will contain the actual device number. The device number must be specified for a release operation since an operational label reassignment may have taken place (e.g., FL keyin).

MRSVP FUNCTIONS

Reserve. If the request is for an unconditional reserve, a message is output to inform the operator of the foreground reserve action (e.g., !!10B-RES, LP02).

If the request is for a conditional reserve, a message is output to inform the operator of the request (e.g., !!10B-REQ, CR03). The operator should then prepare that device for the pending foreground operation, and then reserve the device by an unsolicited key-in of FR (foreground reserve; for example, FR CR03). This will reserve the device for foreground use. If the Reserve Complete

Receiver is specified, it will be entered at this point. Note that the dedicated interrupt location of a task requesting use is indicated as dil- (i.e., !!dil-REQ,LP02).

Release. The peripheral device can be released for background use or the next foreground task by a call to M:RSVP to release the device. The peripheral device specified will be made available for other users or background. A message will be output to inform the operator of the release action if the device is being released to background (e.g., !! REL, CR03). The peripheral device can also be released by an unsolicited key-in of BR (background release). Unsolicited key-ins to reserve and release peripheral devices are described in Chapter 3.

Limitations. The reserve peripheral table will accommodate only as many entries as were specified at SYSGEN, (RSVPTABL, X where X represents the number of entries to be provided for and defaults to 5.

M:DOW (Diagnostic Output Writer or Error Logger – Foreground Only)

M:DOW is a dual-purpose service routine available only to foreground tasks. The function that M:DOW will perform is dependent upon the value of a code in word 0 of the argument list defined in the calling sequence below.

```
LDX     adrlst
RCPYI   P,L
B       M:DOW
```

where adrlst is a pointer to the argument list, the format of which is dependent upon its function code, as shown in Table 10.

Return is always to the location in the L register. The B register is always maintained. If code = 0, status is returned in the E, A, and X registers. This status will be the same as that described for M:READ/M:WRITE. If code ≠ 0, no status will be returned; i.e., E, X, and A registers will be unspecified.

M:DOW FUNCTIONS

Code = 0

Multitask use of the same file may result in a conflict situation wherein a task is unable to output a message because a lower-priority task has control of the file. If such a condition could exist, the higher priority task should call M:DOW, which will wait until end-action-pending occurs, save all status for the lower priority task, and translate the M:DOW argument list to an equivalent M:WRITE call. Since end-action-pending occurs at the I/O interrupt level, this allows multitask use of the same file without affecting low level I/O.

Code ≠ 0

The maintenance of an off-line, dynamic Error Log is valuable in the diagnosis and correction of hardware and hardware/software interface problems. As a SYSGEN option (ERRORLOG), M:DOW is available for such logging purposes. From the user-supplied argument list, M:DOW will create an entry to this log according to Table 10, and will add this entry to the Error Log when RBM becomes active.

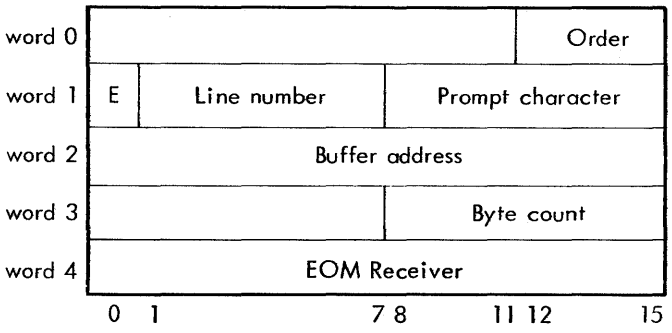
Note: If the Machine Fault Task makes an Error Log entry, interrupts will be effectively inhibited for up to 350 microseconds.

M:COC (Character-Oriented Communications – SYSGEN Optional, Foreground Only)

M:COC performs input, output, and control operations on a specific communication line. The calling sequence is

```
LDX     adrlst      Pointer to the argument list
RCPYI   P,L         Set the return address
B       M:COC       Branch to the routine
```

The argument list pointed to by adrlst is as follows:



where

Order (bits 12–15) is as follows:

Order	Operation
0	Status check of line.
1	Write n bytes, no editing.
2	Read n bytes, no editing.
3	Send break character (long-space).
4	Check previous read or write.
5	Write message of up to n bytes, edited.
6	Read message of up to n bytes, edited.

Table 10. M:DOW Argument Lists

Function	Argument List			
	Word 0/Code[t]	Word 1	Word 2	Word 3
Diagnostic output	0000	oplabel[tt]	Address of buffer	Byte length
	8000	DFN[tt]		
Error Log Entries:				
SIO Failure	0091	DFN[ttt].	-	-
Channel timeout	0092	DFN[ttt]	-	-
Spurious Interrupt	0093	AIO status	Bit 6 = Overflow indicator Bit 7 = Carry indicator	
I/O error	0095	DFN[ttt]	-	-
System startup	0018	-	-	-
Power on	0020	-	-	-
Version	0022	-	-	-
Time stamp	0023	-	-	-
EBCDIC message	0027	-	Address of entry	Byte length
Machine fault	00B1	Fault register contents	-	-
User entry[tttt]	00FF	-	Address of entry	Byte length

[t] Any code other than those indicated in low-order byte is treated as a "no-operation". (The code is shown in hexadecimal.)

[tt] Identifies file/device to be written to.

[ttt] Identifies file/device involved in error (not the error file).

[tttt] User entries receive a time value in words 2 and 3 of the entry.

Order	Operation
7	Disconnect line (turn off data set).
8	Connect line.

where n = 0 < n ≤ 255.

E is 1 if an end-of-message (EOM) receiver is specified; is 0 if no EOM receiver is specified.

Prompt character is meaningful on duplex lines for orders 6 and 8. For order 6, it is the character (EBCDIC) to be output before input is requested. This can be used to signal the operator that input can now begin. For order 8, it specifies the mode in which all communication will be handled on this line until it is disconnected, and it has the following form:

Bit	Value	Meaning
8	1	Echo all input characters.
	0	Do not echo.
9	1	Translate all input from 7-bit ANSCII to EBCDIC, and all output from EBCDIC to ANSCII.
	0	Do not translate any codes.
10	1	Check parity on input and create parity on output (even parity).
	0	Ignore parity.
11-12	00	Device is Model 33/35 teletype.
	01	Device is Model 37 teletype.
	10	Device is keyboard/display.
	11	Device is foreign device, and no echoing, editing, or translation will be performed (overrides setting of bits 8, 9, and 10).
14-15		Communication Lines (for connect order).
	00	Full duplex (echoing accepted).
	01	Simplex – send.
	10	Simplex – receive.
	11	Half-duplex (echoing not accepted).

Note: The time required to turn a half-duplex line from receive to transmit mode is consumed in M:COC at user-program level, not in the interrupt handler, RCOC.

EOM Receiver is used like an AIO Receiver. When an input or output message is completed, the appropriate communications task will branch to the specified EOM receiver address, at the priority level of either the input or output external interrupt, and will show the line number (of the line with the completed message) in the X register. The user program should save this status, trigger an appropriate user interrupt level, and return to the location in the L register. All operations are no-wait operations; that is, the return is immediate upon initiating I/O or performing the connect or status checks. Thus, the EOM receiver is applicable only for read (2 and 6), write (1 and 5), and send break (3) orders. EOM receivers are subject to the same restrictions and precautions as are AIO receivers. (See Chapter 6 for a more detailed discussion of AIO receivers.)

Note: For half-duplex lines the EOM receiver is activated before the EOM sequence is initiated by a subsequent "check" call to M:COC.

Return from M:COC is to the location specified in the L register. On return, the B register remains unchanged; and the E, A, and X registers are set as specified in Tables 11, 12, 13, and 14.

The nine possible orders that can appear in the argument list, and the operation for each, are described below:

0 Check status of line. This operation allows the user to check both the logical condition of the line (line mode, which is one of the unique codes in Table 14) and the physical condition of the line (which is reported just as it is received from the hardware, as shown in Table 13). Only the line number is needed in the argument list.

1 Write n bytes, no editing. If the byte count is odd, the first output transmission takes place from right of the first word, and the left of the first word is ignored. No end-of-message codes are added at the end of the message, and no trailing blanks or null characters are stripped off. Parity generation and translation from EBCDIC to ANSCII are under the control of the specified options for this line.

2 Read n bytes, no editing. A read operation is initiated, with no editing for cancel or character-delete operations, but with a search for any ANSCII control character. Input is terminated if any control character is found or if the specified byte count is exhausted. If any input bytes were received before this read request was given, these bytes are thrown away. The end-of-message character always remains in the user's input buffer, translated to EBCDIC, if specified. The same comments about parity apply for the write operations.

Table 11. Status Returns for M:COC

Operation	Major Status	Action	E	A	X
All operations	Line no. not valid	Return immediately	-1	8	Line no.
	Calling seq. err.		-1	4	Line no.
	Line has disconnected		-1	2	Line no.
	Invalid line status		-1	1	Line no.
Initiate read or write	Line is busy	Return immediately	0	-1	Line no.
	Successfully initiated	Initiate and return	0	0	Line no.
Check previous input or output	Line is busy	Return immediately	0	-1	Line no.
	Operation complete	Clear line and return	0	Completion code	Byte count
Connect or disconnect	Successful connection	Connect and return	0	0	Line no.
Status check	Connected line	Test and return	Line status	Line mode	Line no.

Table 12. Completion Codes

A Register Value	Meaning
0	Successful completion.
1	Parity error on some byte read.
2	Break condition exists.

Table 13. Line Status

E Register Bits	Meaning
0-11	Not used.
12-13	Receiver status 00: Data set not ready (if data sets are used), or receiver not installed. 01: Receiver on. 10: Receiver off. 11: Break (long space) detected.
14-15	Transmitter status 0-: Data set not reporting "clear to send" (if data sets are used), or transmitter not installed. 10: Transmission in progress. 11: Ready to send.

Table 14. Line Mode

A Register Value	Meaning
0	Line disconnected.
1	Output mode.
2	Prompt character output (then switch to input).
3	Input mode.
4	Inactive mode.
5	Message complete.
6	EOM sequence initiated (half-duplex only).

3 Send break character (long-space). If the line is in an inactive mode, the long-space is sent immediately. If the line is in a write mode or a read mode, the operation is terminated and the long-space is then sent. In the argument list, only the line number is meaningful.

4 Check previous read or write. This operation is required for all read and write operations, whether or not an EOM receiver is specified. The user buffer remains busy until the previous operation is checked. The line is then set inactive and becomes ready for subsequent use. This is the only way to determine break conditions. The return status is shown in Tables 11 and 12. Only the line number is meaningful in the argument list.

5 Write message of up to n bytes, edited. This operates like the write operation without editing except (1) that trailing blanks and trailing null characters are removed and (2) that appropriate control characters are added as the final characters of the message.

6 Read message of up to n bytes, edited. This operates like the read without editing, except that ignore, backspace, and cancel operations are in effect for the current line; when any of these special characters are encountered, the proper effect takes place on the line and the user's buffer is modified accordingly. (Note that the backspace is an editing, or destructive, backspace; that is, the previous character is deleted from the user's buffer.) The prompt character, if nonzero, is output prior to the read operation. (See Table 15 for a summary of editing operations.)

7 Disconnect line. The send and/or receive modules of the line are turned off, the data set is disconnected, and the logical status is set to disconnect.

8 Connect line. The communication mode option for the line, simplex or duplex, is matched against the physical structure of the line and, where appropriate, the receiver is turned on. Mode conflicts are returned as invalid line status. The logical line mode is set to "inactive" and the other options are set. The connected line is assumed to be a dedicated line or a line that has already dialed-in. A user program can poll the lines with a "check status" order prior to logical connection to determine when a line has been physically connected (i.e., data sets ready).

FUNCTIONAL DESCRIPTION OF COC PACKAGE

The COC software package manages character-oriented telecommunications equipment (normally Teletype-compatible devices) at the message level, providing translation, echoing, parity checking, and the line editing as required. It consists of two portions, M:COC and RCOC.

M:COC. This is a monitor service routine that performs all control operations and initiates all reads and writes. It is part of the nonresident RBM overlay structure.

RCOC. This is a resident foreground program, usually requiring installation modifications, that consists of the following tasks and related items:

1. An initialization routine.

2. An input-interrupt handler connected to the input interrupt of the COC controller (7611), which translates and edits input characters, echoes the characters as required, and forms input messages.

3. An output-interrupt handler connected to the output interrupt of the 7611, which translates and transmits output characters and performs line formatting at end-of-message (character-count completion only).

Table 15. Summary of Editing Operations

Operation	Codes Used		
	33/35	37	Character Display
User-generated end-of-message character on input, edited	CR or LF or BREAK	NL or BREAK	NL or INTERRUPT
System-generated end-of-message character on input	LF or CR (opposite of user input); CR and LF on BREAK	None for NL; NL for BREAK	None for NL; NL for INTERRUPT
Attention code; used to terminate input or output	BREAK	BREAK	INTERRUPT
Ignore this character, except after ESC	RUBOUT or ESC,SPACE	DEL or ESC,SPACE	DEL or ESC,SPACE
System-generated characters on output at end-of-message	CR,LF,RUBOUT	NL,RUBOUT	NL,5 - NULL
Delete previous character	ESC,RUBOUT (echo ←)	ESC,DELETE (echo \)	ESC,DELETE or EM operation
Delete current line	ESC,X	ESC,X	ESC,X or CR,CAN

4. Input and output translation tables (ANSCII to EBCDIC and vice versa).

5. A circular input buffer (i.e., "ring buffer"), which overlays the initialization routine (item 1).

RCOC may be loaded at system boot time or as needed (installation option). When loaded, the initialization routine automatically connects the COC handler tasks to their respective interrupts, establishes linkage for M:COC, initializes the COC for input, and exits. At this point, all lines are set to the (logically) disconnected status, ready to be tested, connected, and used via calls to M:COC.

M:COC FUNCTIONS

All line-control and read-write operations are initiated by means of user calls to M:COC. Once RCOC has been initialized, all input/output requests are rejected by M:COC until the line is connected. If a line is dedicated (i.e., leased or "hardwired") or if a dial-up line has dialed in, only a connect (order 8) call to M:COC is required. If the line is to be "dialed out" (physical activation from the computer end rather than from the terminal end), an M:IOEX SIO-order call to the Automatic Dialing Equipment must precede the M:COC connect request for each line (see "Automatic Dialing" below for further details).

A check-line-status (order 0) call may be issued prior to a connect request to check for line-operational and physically-activated conditions, in which case detailed line-status and line-mode information is returned (Tables 13 and 14). If this is not done and a connect request is issued for a line that is nonexistant or nonoperational, i.e., no send and/or receive module installed or receive module will not turn on (but whose line number is valid), the following operator's message is issued and an invalid-line-status major status is returned:

TROUBLE LINE nn

If the line is not physically activated, e.g., not dialed-in (data set not ready or not "clear to send"), invalid-line-status is returned also. If the specified line number is not a valid one, this is so reported. (The range of valid line numbers is determined during the assembly of RCOC.) See Table 11 for major-status returns.

A successful connect call for a given line sets the logical line mode to "inactive", in which mode any input received on the line is ignored, but the line is available for I/O requests. Subsequent I/O operations on that line must be initiated sequentially with a check-previous-operation (order 4) call intervening between successive read/write calls (I/O requests are not queued). As each read or write operation is completed, the logical line mode is set to "message complete". At this point the line is still busy and can only be cleared (set to inactive) by the check-previous-operation call. (This call, order 4, is not required after a check-status, connect, or disconnect request.) The check-status (order 0) call may be executed at any time.

Program and Interrupt-Task Relationship. A read request simply sets the line mode to "input" at calling program level, which in turn causes the input interrupt task to accept input on that line and build the input message in the user's buffer, all at interrupt level. A write request sets the line mode to "output" and causes M:COC to transmit the first character in the user's buffer at calling program level. Thereafter, the output interrupt task automatically transmits the remaining characters, one per COC output interrupt (i.e., one each "output word time") until the entire message is sent, all at interrupt level.

As each input or output message is completed (or otherwise terminated), the line is set to "message complete", line mode 5, and the user's EOM receiver (if present) is executed at the interrupt level. Normally, the receiver should trigger the requesting task and (always) return via register L.

AUTOMATIC DIALING

If Automatic Dialing Equipment (ADE) is included, real-time tasks can dial a terminal and connect it to a predetermined COC line for that terminal. The ADE is a multiunit controller that controls up to 16 dial positions and corresponding lines. It is connected to a dedicated IOP channel (additional to the COC's).

The dialing operation can be accomplished via M:IOEX. A TDV order should first be issued to ensure that the dial position is available. Then an SIO order can be issued to activate the ADE and address the dial position. Any order byte will be interpreted as a "write". The memory buffer contains the number of the data set being dialed (two digits per word, each digit occupying the rightmost four bits of the byte in four-bit BCD). After the dialing procedure has been completed, the task should check the status of the COC line before attempting to send or write on it.

5. I/O OPERATIONS

BYTE-ORIENTED SYSTEM

The Monitor performs all I/O services for the byte-oriented I/O system. This includes:

- Logical-to-physical device equivalencing.
- Initiating I/O requests.
- Standard error checking and retry (optional).
- Task dismissal on "wait" I/O (optional).
- Software checking of background requests to preserve protection of foreground and Monitor.
- Optionally generating device order bytes for device-independent operations.
- Accepting user-generated IOCDs and device order bytes to provide complete control for a user's program.
- Using data chaining for foreground programs performing scatter-read or gather-write operations.
- Reading or punching cards in either BCD or EBCDIC.
- Positioning magnetic tapes and RAD files.
- Editing from paper tape or keyboard/printer.
- All I/O interrupt handling.
- Managing both temporary and permanent RAD files.
- Limiting channel active time for I/O transfers.

I/O INITIATION

Whenever a task needs to initiate an I/O operation, it calls on the appropriate Monitor I/O routine (see Chapter 4 for complete calling sequences). These Monitor I/O routines are reentrant, so that a higher priority task may interrupt and request I/O during the initiation of a lower-priority task, in which case the low-priority task is suspended and the higher-priority task satisfied first.

A real-time foreground program may acquire control of a multidevice controller from background users at the completion of any current I/O. This technique is used in place of queuing. All Monitor I/O initiation is made at the priority of the calling task, with background tasks having the lowest priority.

The channel time limits imposed by the Monitor on standard devices are as follows:

Device Type	Maximum Allowable Channel Active Time (seconds)
KP	255
LP	3
CR	3
CP	3
MT (9 track)	10
PT	# chars. x rate
MT (7 track)	10
RD 7202/04	3
RD 7242/46	4
RD 7251/52	3
PL	Not imposed
LD (logical device)	Not imposed

END ACTION

The chapter on Operator Communication specifies the possible error messages. Generally, standard error recovery takes place when the I/O is checked for completion rather than on the I/O interrupt. This means that error recovery for the background will be processed at the priority level of the background rather than at the I/O interrupt priority level. However, there is a provision for the real-time foreground user to specify an end-action routine to be called when the Monitor answers the I/O interrupt. This is the AIO receiver address in the I/O calling sequence, and it is to be used only when more sophisticated end-action is required or when a foreground task is to be restored to active status at channel end. The routine is processed at the priority level of the I/O interrupt, so the processing should be of very short duration. Reentrancy in this routine is the user's responsibility. For example, this routine might consist of storing the I/O status information and then triggering a lower-level external interrupt through a Write Direct, where this lower-level task performs the actual processing. The end-action routine should then return to the task from which it originally came (by RCPY L,P).

The form of the call to the AIO receiver is

```
LDA     aiodsb
RCPYI   P, L
B       AIO receiver address
```

(device status byte from AIO in bits 0-7; device number in bits 8-15)

The AIO Receiver routine should return to the location contained in the L register on the entry. All registers are assumed to be volatile, which means that they need not be saved and restored to their former contents.

The purpose of the AIO Receiver technique is to allow a real-time user program to be informed by RBM when channel end occurs on a particular I/O operation. It is used instead of I/O queueing by the Monitor. Typically a foreground program wishing to maximize I/O and computation overlap will issue an I/O request with the no-wait option and with an AIO Receiver address specified. When the I/O is successfully initiated, the foreground task exits from the active state (by a call to M:EXIT) and is restored to active status at channel end by a Write Direct to trigger the interrupt level from the AIO Receiver. The foreground program must then return to the Monitor I/O routine with the "check" option to complete the end action on the file. See Chapter 6 for a more detailed discussion of AIO Receivers.

Note: For transfers invoking blocked files where no I/O is actually performed, the X register will contain -1 to indicate that the AIO receiver will not be entered.

LOGICAL/PHYSICAL DEVICE EQUIVALENCE

When writing a foreground or background program in either Extended Symbol or FORTRAN, the user is not required to know the actual physical device number that will be used in the input/output operation. Two ways are provided under RBM to help the user select the input/output device on a logical rather than physical basis.

The first method is the direct logical reference. The user can specify a device-file number in his calling parameters to the input/output routines, and RBM will translate this into an actual physical device number. There may be several device-file numbers pointing to the same physical device; however, only one device-file number is generally needed per device per active task in the system. Each device-file number can be used by only one task at a time. This is a necessary restriction since the I/O status is saved in the device-file number table in the RBM and independent operation by several tasks on the same device would cause invalid status from the separate tasks using it.

The second method is device referencing through indirect logical reference. This method first assigns a device unit number or an operational label to a device-file number, which in turn is assigned to a physical device number. The equivalence of operational labels or device unit numbers and the device-file numbers is set at System Generation time for certain standard devices, as shown in Tables 2 and 16. The standard assignments may be changed later by use of !ASSIGN or !DEFINE control commands.

Table 16. Standard Device Unit Numbers

Device Unit Number	Standard Assignment
101	Keyboard/printer input
102	Keyboard/printer output
103	Paper tape reader
104	Paper tape punch
105	Card reader
106	Card punch
108	Line printer

Table 2 shows the standard background operational labels. The devices and functions shown indicate how the standard processors use these labels. Since each I/O call must specify a byte count, a user program can read any number of bytes from SI (if SI is magnetic tape, for example). The labels are merely a name. There is no restriction on the record size except as imposed by the peripheral devices.

LOGICAL DEVICES

In addition to the foregoing use of the term "logical device," "Logical Device" is also used to refer to a SYSGEN mechanism for reserving logical groups of DFNs for a combination of foreground and background use to accomplish information or data transmission between tasks without the use of any real physical device. (Refer to the RBM System Management Reference Manual for a description of the SYSGEN mechanism for defining a Logical Device in this sense.)

Logical Devices are defined at SYSGEN via a 2-character mnemonic[†] (for model number), and an accompanying pseudo-device number (indicating a channel number, preferably unique). The user performs reads or writes on DFNs (or assigned oplabels) associated with the LDs via calls on M:READ and M:WRITE. Two DFNs must be assigned to define one Logical Device. Communication between foreground and background tasks is accomplished by use of the foreground (F)/background (B) SYSGEN option at definition of the LD.

One example of possible use would be where a task receives data from a hardware device via standard oplabel or DFN. This data may be manipulated (if desired) by the task and passed on to another task via a pair of DFNs associated with the same LD. The receiving task may pass the data to a different LD or to a real physical device.

[†]The mnemonic "LD" or any other 2-character mnemonic other than RD or XX can be used. This mnemonic may indicate a device type the Logical Device is to represent; e.g., LP for Line Printer as required by the printer symbiont.

There are no restrictions as to direction of flow of information. Either DFN associated with an LD may be used to read or write to the other DFN associated with the same LD. Two DFNs must be associated with one pseudo-device number to define an LD.

When using an LD, an I/O operation takes place between the two DFNs associated with the LD. That is, an I/O operation is only satisfied if a read/write pair of operations occurs within the definition of one LD. If a task communicates with more that one LD, another task (or tasks) must perform the reciprocal I/O operation on the DFN of each of the LDs the first task performed I/O on. A pre-I/O edit routine for LDs satisfies the I/O operation only when each of the reciprocal I/O requests have been made against an LD. Refer to the RBM Technical Manual for further discussion of the LD mechanism.

RAD FILES

The two types of RAD files available are sequential files and random files. A sequential file may be used like a single-file magnetic tape, whereas a random file may be used like a truly direct-access device. The capabilities and restrictions of each type of file are described below.

Random and sequential files vary in two primary respects:

1. Sequential files cannot be accessed randomly; the next record to be accessed is the one at which the file happens to be positioned.

2. Sequential files can only be updated at the end.

Random and sequential files share the following attributes:

1. Both are available to foreground and background tasks (but not concurrently).

2. Both are available to routines M:READ, M:WRITE, and M:CTRL, but not to M:IOEX.

3. Both can be blocked. The Monitor I/O routines do the blocking and unblocking.

4. Logical records may be less than, equal to, or greater than the RAD sector size. Unblocked records always start on a sector boundary. Therefore, if a logical record is less than a RAD sector and is unblocked, the remaining bytes of the sector will be ignored. If a logical record is greater than a sector, it will occupy an integral number of physical sectors and the remaining bytes of the last sector will be ignored.

5. BOT (beginning-of-tape) is defined as the logical load-point and equals the first sector of the file. EOT is defined as the logical end-of-tape and equals the last sector +1 of the file. EOF (end-of-file) is defined as the logical file mark (which may or may not exist).

6. Both can be positioned by !REWIND, !FBACK, and !FSKIP commands.

7. Foreground I/O requests can specify an AIO Receiver at channel end for physical I/O transfers. When operations involve only logical I/O transfers, the AIO Receiver will be ignored. A flag will be set (x = -1) indicating the AIO Receiver is not to be acknowledged, (see M:READ/M:WRITE status returns).

8. Operational labels can be equated to permanent files on the RAD, or be allocated from available temporary RAD space. This can be accomplished either through control cards or through Monitor service calls at execution time.

9. When the operational label is defined or assigned to a permanent file, it is automatically positioned at the BOT.

10. The transfer of any even number of bytes (to a maximum of 65,534) may be requested, provided that the transfer will not extend past the file boundary for unblocked files. For blocked files a single record is processed on each call.

SEQUENTIAL FILES

1. Sequential RAD files can be compressed (with blanks removed) if they are EBCDIC data. The Monitor I/O routines do the compressing and expanding but do not check for binary data. Compressed records are always blocked and of variable size; therefore the logical record size has no meaning except when allocating the file.

2. As on magnetic tape, once a logical record or file mark is written on a file, any records or filemarks previously written beyond that point are unpredictable.

3. Sequential RAD files (except compressed files) can be spaced forward or backward by logical records. Selected records may be read from a blocked sequential file

by spacing records forward or backward, but only records at the end of a sequential file should be written, i.e., update in-place not permitted.

4. As on magnetic tape, the only record that can be written at the EOT is the logical file mark.

RANDOM FILES

1. All unblocked I/O transfers start on a granule boundary within a file. These granule boundaries are addressed as a number that represents the displacement of the granule from the start of the file, beginning with zero. A granule boundary always begins on a sector boundary but need not end on one (see discussion of granules below).

2. When a random file is defined, the user may specify a FORTRAN logical record size and a pointer to the word where the last referenced FORTRAN logical record +1 is stored. This information, although unused by the Monitor, is stored in the file and may be requested by executing programs or processors (such as the FORTRAN compiler), if necessary.

3. Random files may not be compressed. They may be blocked with transfer on a logical record basis. In this case, the Monitor performs all blocking/deblocking operations. Any Write operations are really an update in place and unmodified portions of a block are preserved. A block is not read into core if it is already in core from a previous operation.

4. EOF has no meaning in random files except for file saving, truncating, and mapping purposes.

5. Random files (either blocked or unblocked) may be accessed sequentially or randomly. At the end of any operation, RBM automatically updates the record displacement pointer to the "next" record. The pointer can be "set" by any random operation, is initially set to the beginning of the file, and may be changed by M:CTRL.

   As much data as specified by the byte count will be transferred for the unblocked random files but only one record at a time will be transferred for blocked random files and incorrect length can occur.

GRANULES

While a granule is usually synonymous with a sector on a device, it may be defined (on a file basis) to be equivalent to any of the following:

- A partial sector.
- One sector.
- Several sectors.

A granule always begins on a sector boundary but need not end on such a boundary. For example, to make the 7204 RAD and the 7242 disk pack transfers equivalent, a granule can be defined to be 1024 bytes; this is then one sector on the disk pack and two sectors plus a fraction of a sector on the 7204 RAD.

BLOCKING BUFFERS

The RBM system allows for automatic assignment of blocking buffers for files of blocked, compressed, or packed format. The number of buffers required by a program may be specified through the !$BLOCK control card of the Overlay Loader. Such buffers as will fit within unused memory (UMEM) of the loaded program may be allocated to a maximum of 16. Size of these blocks is determined by the value of K:BLOCK. Use of such blocks is identified and maintained in the task TCB (use bits). Assignment is made from this pool of buffers as required explicitly through an M:OPEN call or implicitly through the first use of the file. Closing a file that uses a block from the pool will free its buffer for later assignment. Thus, a minimum requirement for pool buffers may be achieved through a judicious opening and closing of files requiring such blocks.

The !$BUFEND control command in conjunction with the !$BLOCK command will allow the foreground user to allocate an area outside his program as the buffer pool.

The ability of more than one file to share the same buffer block is provided to accommodate "packed" files whose records may be accessed randomly and thus may require a fresh block with each call to M:READ/WRITE. The capability of sharing a single buffer from the program buffer pool is conveyed by the !$BLOCK command as the program load file is generated. This capability is registered in the TCB as the program is loaded into memory and the first buffer from the pool is identified as the sharable block. Packed random files may be individually identified as accepting a shared buffer through an ASSIGN or DEFINE parameter. Subsequent operations with such a file must be on a "wait" basis since the shared buffer is freed before completing each read or write request. If the transfer request is not I/O with wait, a calling sequence error will be returned. This is true whether the buffer is from the buffer block pool or whether it is allocated explicitly (M:OPEN) within the program.

A buffer block may not be shared by other than packed random files of the same task.

Records of compressed, blocked and packed files are treated as a contiguous stream of data for blocking purposes. As such, individual records may overlap block boundaries without concern to blocking procedures.

RAD FILE MANAGEMENT

RBM permits allocation of the RAD into the subsections shown in Figure 4. The exact bounds on these sections are computed from the size of required contents or selected by the user in accordance with the anticipated use of the system. In either case, the bounds are set during System Generation, and cannot be changed except by a new System Generation. RBM maintains directories for as many areas as the user specifies up to 35, plus: the Checkpoint area, the Background temp area, the System Library, System Processor area, and System Data area. RBM also maintains control of the checkpoint area. The background temporary space is allocated from control command inputs or from calls to M:DEFINE as requested.

Areas need not be allocated contiguously (RAD tracks may be skipped between areas), and can be distributed over more than one RAD. One to 16 areas may be allocated on each RAD or disk pack. However, each area must exist entirely on a single RAD. If there is more than one RAD on the system, one will be designated as the RBM System RAD, which will receive any default areas. Any RAD with sector 0 available will receive a bootstrap in that area.

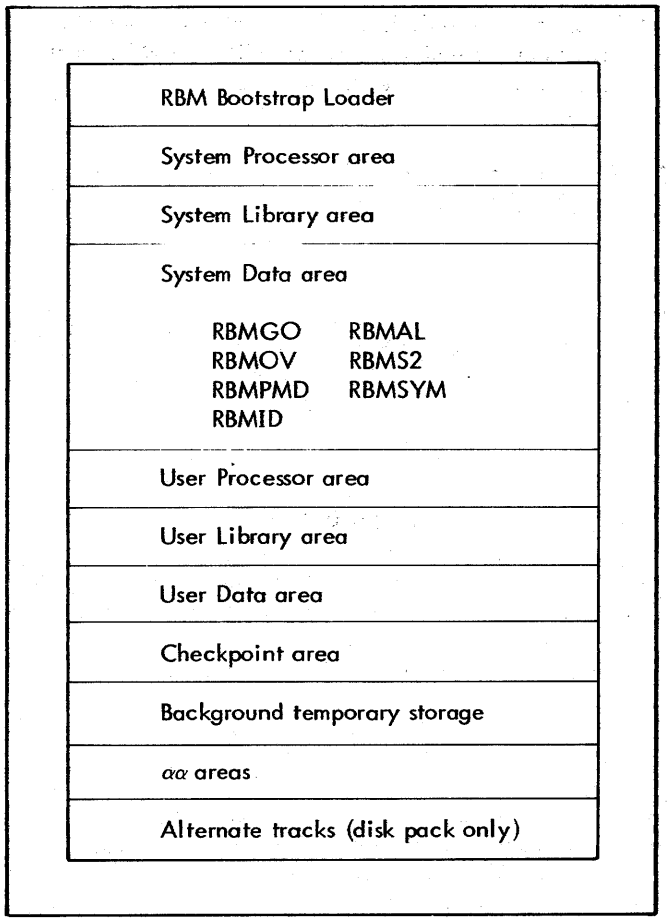


Figure 4. RAD Allocation

6. REAL-TIME PROGRAMMING

FOREGROUND PROGRAMS

A foreground program is one that operates in protected memory, utilizes foreground operational labels or device unit numbers, and has access to privileged instructions. It is protected from any background interference through an integrated hardware/software protection scheme. A foreground program may be classified as either a resident foreground program, a semiresident foreground program, or a nonresident foreground program, and it is important that this distinction be understood.

RESIDENT FOREGROUND PROGRAMS

Foreground programs are defined as resident through the RAD Editor when their files are created on the user processor area of the RAD. They are loaded into core from the RAD whenever the RBM system is booted, and are either automatically armed, enabled and (optionally) triggered, or they initialize themselves through their own initialization routines. Once loaded into core for execution, resident foreground programs remain resident until the RBM system is again booted from the RAD.

SEMIRESIDENT FOREGROUND PROGRAMS

Semiresident foreground programs are normally not in core memory. They are not read into core when the RBM system is booted but must be called in explicitly when needed. Semiresident foreground programs, when loaded, reside in the resident foreground area. The user must schedule the loading of semiresident foreground programs because the Monitor provides no protection against overlay or overloading. When loaded, they may be automatically armed, enabled and (optionally) triggered, or they may initialize themselves through their own initialization routines.

NONRESIDENT FOREGROUND PROGRAMS

Nonresident foreground programs are normally not in core memory. They are not read into core when the RBM system is booted but must be called in explicitly when needed. Nonresident foreground programs, when loaded, reside in the nonresident foreground area, and the area is then considered "active" and is not available for subsequent use by other programs (including the Monitor) until the program occupying this area releases it by "unloading". This feature is useful when a system has several nonresident foreground programs that have a resource allocation problem or are connected to the same interrupt level. The Monitor will control access to the nonresident foreground area, thus providing protection against multiple loading of these conflicting programs.

If nonresident programs are to be used, at least K:BLOCK+17 memory locations must be allocated for the nonresident foreground area of core. If allocated, the nonresident foreground area is adjacent to the background. If a nonresident foreground program is to be loaded and the length of the longest path (including COMMON) exceeds the size of the nonresident foreground area, the background is automatically checkpointed to allow the program to extend to the background. The background remains checkpointed until the nonresident foreground program unloads by a call to M:LOAD. When loaded, nonresident foreground programs may be automatically armed, enabled and (optionally) triggered; or they may initialize themselves through their own initialization routines.

MONITOR TASKS

The relative priorities of the separate Monitor tasks are given in descending order below:

Highest Counters (optional).

Power On Task.

Power Off Task.

Machine Fault Task.

Protection Violation Task (optional).

Multiply Exception Task (optional)[†].

Divide Exception Task (optional)[†].

Real-time tasks, if any, higher than I/O level.

Input/Output Task.

Control Panel Task.

Counters = 0 (optional).

Real-time tasks, if any, lower than I/O level.

RBM Control Task (lowest hardware level).

Background "tasks", lower than all hardware levels.

Although the tasks are not reentrant, they are serially reusable; that is, as soon as a task finishes processing one request, it can immediately process another. For example, I/O interrupts are processed one at a time, with the highest priority device always processed first if several interrupts are waiting, but as soon as the processing of one interrupt request has been completed, another request for a separate device can be processed.

POWER ON TASK

The Power On Task performs the following operations:

- Waits for acceptable RAD status.
- Arms and enables all RBM interrupts.

[†]Sigma 2/3 only.

- Triggers the RBM Control Task to send a !!POWER ON message.
- Restores protection registers to failure-time contents.
- Loads and links to the Power-On receiver if specified in pointer location X'1A9'. If the computer is a Sigma 3, the X register will point to the interrupt status saved at Power-Off time. This data is organized as follows, one bit per interrupt per word as described for the WD-instruction register bits in the appropriate computer reference manual:

  0, 1: 0 if recovery will be attempted, 3 if recovery will not be attempted.

  1, 1: Group '0' interrupts enabled.

  2, 1: Group '0' interrupts armed or waiting.

  3, 1: Group '0' interrupts waiting or active.

  If the computer is a Model 530, the X register will point to a data area organized as follows:

  0, 1: 0 (Recovery will be attempted).

  1, 1: Group '0' interrupts enabled.

  2, 1: Group '5' interrupts enabled.

  3, 1: Group '6' interrupts enabled.

  4, 1: Group '0' interrupts armed or waiting.

  5, 1: Group '5' interrupts armed or waiting.

  6, 1: Group '6' interrupts armed or waiting.

  7, 1: Group '0' interrupts waiting or active.

  8, 1: Group '5' interrupts waiting or active.

  9, 1: Group '6' interrupts waiting or active.

- Restores status at Power-Off time and exits if the computer is a Sigma 3 with no external interrupts, or is a Model 530.
- Restores context and exits if it is a Sigma 2 or there are external interrupts and background is active.

If none of the above conditions are satisfied, the Power-On interrupt is cleared and a SYSERR is forced.

Since Power-On processing is installation dependent and correct recovery cannot always be guaranteed, a user-developed Power-On Receiver may be used to restart after a power failure. The following action may be taken within the receiver:

1. Timeout errors will be simulated on all active I/O channels at Power-Off time. Code within the receivers may restart I/O for these devices.
2. The interrupt status is determined, in general, through the TCB chain (each TCB contains the address of the TCB of the task it interrupted). Race conditions can exist that may cause this chain to inaccurately reflect the interrupt status, although the PSD chain is correct. If this risk is considered negligible or the effects unharmful, the tasks can be reactivated through the TCB chain by the receiver.
3. The foreground Power-On Receiver may activate one or more foreground tasks or take other special action to restart the system. This may involve going to some recent checkpoint.
4. The receiver may exist from the Power-On routine by going to M:EXIT.

POWER OFF TASK

The Power Off Task performs the following operations:

- Saves all available interrupt status, depending on model of CPU (see above).
- Saves context via a call to M:SAVE.
- Scans the Channel Status Table and issues an HIO to any channel flagged active and saves the device status byte and the even and odd channel register contents in the File Control Table.
- Interrogates location X'1A8' for a Power-Off receiver. If one is specified, a branch is made to it; otherwise, the Power Off Task waits for the power-on interrupt.

MACHINE FAULT TASK

For Sigma 2/3 Systems: The Machine Fault task responds to the following Sigma 2/3 machine fault conditions, listed in order of priority:

1. Memory parity error.
2. External IOP timeout.
3. Incorrect direct I/O.
4. Internal IOP timeout.
5. Combination of conditions 2 or 3 and 4.

} Sigma 3 only.

Of these conditions, background can only cause a memory parity error. When this occurs, the Machine Fault task triggers RBM and the background task is aborted with an error code of PE. For all of the above conditions, including parity error when background is not active, an appropriate foreground receiver will be tested, as specified below. If this receiver pointer is zero, the action specified below will be taken. Otherwise, the receiver will be linked to via a RCPYI P, L. If the receiver returns, the Machine Fault task will proceed as if a receiver was not specified. The receiver may correct the situation and simply call M:EXIT.

Condition	Receiver Pointer Address	Action
1	X '1AD'	Abort, code = PE
2	X '1AB'	SYSERR, code = ET
3	X '1AA'	Abort, code = MF
4	X '1AC'	Machine Fault Message

Abort action consists of disabling the associated interrupt and exiting the task. If the task occupies the nonresident area, an UNLOAD will be performed. If an IIOP timeout occurs, RBM will be triggered to write the "Machine Fault..." message. The active task will not be terminated but, on exit from the Machine Fault task, overflow and carry will be set to indicate device not recognized.

All foreground abort messages and the "Machine Fault..." message will be written at the RBM Control Task level. Therefore, if two consecutive foreground tasks abort, only the message for the lower priority task will appear. However, both a foreground abort message and the "Machine Fault..." message may accumulate.

For Model 530 Systems: The Machine Fault Task responds to all Model 530 hardware-detected machine faults, as reported by the fault register. If the error-logging option has been selected at SYSGEN, the contents of the fault register is logged in the system error file (ERRFILE, SD) along with pertinent context data.

The fault condition is analyzed and a severity code (0 through 4) is generated according to the breakdown shown in Table 17. Location X'1AA' is interrogated for a machine-fault receiver pointer. If the pointer is nonzero, control is transferred to the receiver with the X and L registers set as follows:

Register X – pointer to a two-word field, where word 0 contains the severity-level code (0-4), and word 1 contains the fault code (current fault-register content, see Table 17).

Register L – return location in Machine Fault Task.

If the receiver pointer is zero, or if the receiver returns control (via register L), the Machine Fault Task will log the error and proceed, as summarized below, according to the assigned severity level:

Severity	Action
0	Immediate exit, i.e., log fault condition only.
1	Retry instruction at which fault occurred.
2	Abort task causing the fault with abort-code MF.
3.	Transfer to SYSERR routine, with code MF, which writes SYSERR message and brings system to orderly halt.
4	Immediate system halt with code MF in register A, fault code in register X.

(The receiver can change the severity level and return, with appropriate effect.)

Note that, other than error logging, no action is taken on multiple fault conditions (severity 0).

Table 17. Machine Fault Classification by Severity Levels

Severity Code and Meaning	Fault Classification	Fault Register Contents (Hex.)
0 – Error logging only.	All faults not listed below, including any combination of multiple faults (i.e., faults reported concurrently).	Any other than below.
1 – Retryable: not second consecutive occurrence at same location.	DIO data-in parity error.	8210
	DFSA constantly high.	8204
	Address-parity error on instruction fetch.	81>2
	Data-parity error on instruction fetch.	81>1
	DIO argument field error.	n008
	DIO data-out parity error.	n004
	NIO data parity error.	n001
	Interrupt-Master fault.	0802
2 – Serious: not retryable but limited to a specific task.	No DFSA response.	8208
	Unimplemented instruction.[t]	8202
	Memory module absent (i.e., nonexistent address).[tt]	81x4
	Address-parity error on operand fetch.	81<2
	Data-parity error on operand fetch.	81<1
	PCP pseudo-fault (TRACE on and PCP interrupt).	0804
	DIO fault: parity error on external write direct.	0801

Table 17. Machine Fault Classification by Severity Levels (cont.)

Severity Code and Meaning	Fault Classification	Fault Register Contents (Hex.)
3 – Critical: affects entire system.	IOP fault.	n020
	Watchdog timeout (special system devices).	n010
4 – Catastrophic: the integrity of any operation cannot be assured.	All mode-3 CPU faults: Instruction timing error. ROS address-parity error. ROS data-parity error. MCM error. AU parity error.	83xx
	Control module error.	8220

Legend: < indicates less than 8, > indicates greater than 7, n = 1 for IOP1; 2 for IOP2, and x indicates nonspecified value.

†The error receiver is not entered if an unimplemented instruction is executed by a background program.

††The error receiver is not entered if a nonexistent memory address is referenced by a background program (abort with code NA).

PROTECTION VIOLATION TASK

Any attempt by the background to modify the contents of protected memory, or to execute a privileged instruction, will cause the Protection Violation Task to abort the background program, using the same method as the Machine Fault Task.

Unavailable core is set protected. On Sigma 2/3 systems, write attempts to unavailable core cause protection errors, and read attempts from unavailable core cause parity errors. The abort code after a protection error shows the location causing the error if the error was an invalid store or a privileged instruction. An attempt by the background to branch to protected memory will cause an abort with the address of the location that was being branched to. Note that Monitor service routine calls actually cause a protection violation from the background. However, if the branch address and the return to the background are valid, the branch is permitted.

The set multiple precision mode instruction, RD X'81', does not cause a protection violation when multiple precision hardware is implemented.

MULTIPLY/DIVIDE EXCEPTION TASKS

Sigma 2/3 Systems Only: These tasks simulate a Multiply or Divide instruction for Sigma 2/3 computers not equipped with Multiply/Divide hardware. They are not reentrant, and all lower interrupts are essentially locked out for the duration of the simulation (approximately 250 to 300 CPU microseconds.)

INPUT/OUTPUT TASK

After an input/output interrupt, the Input/Output Task identifies the highest priority device with a pending interrupt. It then clears the channel activity status and sets the operational status byte count residue in the proper device-file status table, if the device is no longer operating. (The channel is not cleared for a zero-byte-count interrupt.) If a foreground AIO Receiver was specified (for a description of an AIO Receiver, see "I/O Operations" in Chapter 5), control is transferred to this receiver at the I/O priority level. It is expected that the AIO Receiver exits properly.

To minimize interrupt inhibit time, the channel registers are loaded and the I/O initiating SIO is issued at the I/O interrupt priority level. Consequently, any task with a priority level higher than I/O must not use M:READ, M:WRITE, or M:IOEX to perform I/O, but may perform its own I/O without use of the I/O interrupt.

When Clock 1 is employed (a SYSGEN option), M:READ/M:WRITE operations are subject to a time limit. Clock 1 is used to ensure that no channel is active beyond a preset limit. If the limit is exceeded, an HIO is issued to the offending device and appropriate end action will be taken.

CONTROL PANEL TASK

A Control Panel Interrupt causes the Control Panel Task to perform one of two functions: (1) remove the foreground task and (2) notify the RBM Control Task of a pending key-in. If the Control Panel data switches are set appropriately,

a foreground disable and abort may occur (see "Operator Control", Chapter 3). Otherwise, the Control Panel Task sets the key-in flag for the RBM Control Task, triggers the RBM Control Task and exits. The key-in operation itself is performed at the level of the RBM Control Task.

RBM CONTROL TASK

This task controls unsolicited key-ins and background operations. It is the only RBM task that actually performs input/output and, therefore, is the only task that requires temporary stack space for the reentrant RBM input/output routines.

SCHEDULING RESIDENT FOREGROUND TASKS

When several programs and tasks are simultaneously located in core memory, scheduling is required for the orderly transfer of control from one task to another. Scheduling takes place in accordance with the following rules:

1. When no background or foreground task is active in the system, the Monitor enters the "idle" state until the operator directs the loading of a set of control commands from an input device.

2. After a background program is loaded, the Monitor transfers control to the program by an exit sequence from the RBM Control Task. During execution of the background program (if the program is waiting for its own I/O to complete), there can be nothing else in execution in the system. That is, the Monitor makes no attempt to multiprogram to absorb idle time. If there is an armed and enabled resident foreground task in core, the foreground program may receive an interrupt from some external source.

3. After entry, the interrupting task saves the contents of any registers it will alter and proceeds to carry out its function. The task may use either the M:SAVE service routine to perform the saving opertions or it may save the contents of the registers itself.

4. When the real-time task is completed, it may restore the context of the interrupted task and exit via the standard interrupt exit procedure or may have these functions performed by the M:EXIT service routine.

   Warning: If the real-time task has changed the state of the interrupt levels by arming or disarming any active interrupt, the system integrity is lost. The enable/disable feature should be used to prevent interrupts until an orderly exit and inactive state is achieved.

Note that this is a last-in, first-out form of scheduling. The interrupting task may itself be interrupted at any time during execution by a higher priority task, up to the maximum possible number of tasks in the system.

Each time, a new task saves the status and register contents of the interrupted task. When the new task exits, control is returned automatically to the task it interrupted. If there is another interrupt waiting between the level of the current task (which is just completing) and the interrupted task, the originally interrupted task is immediately interrupted again and the new (intermediate) task follows the same procedure. Thus, it is never necessary for any task to know what task precedes or follows it. The task merely preserves and restores the environment according to the established rules.

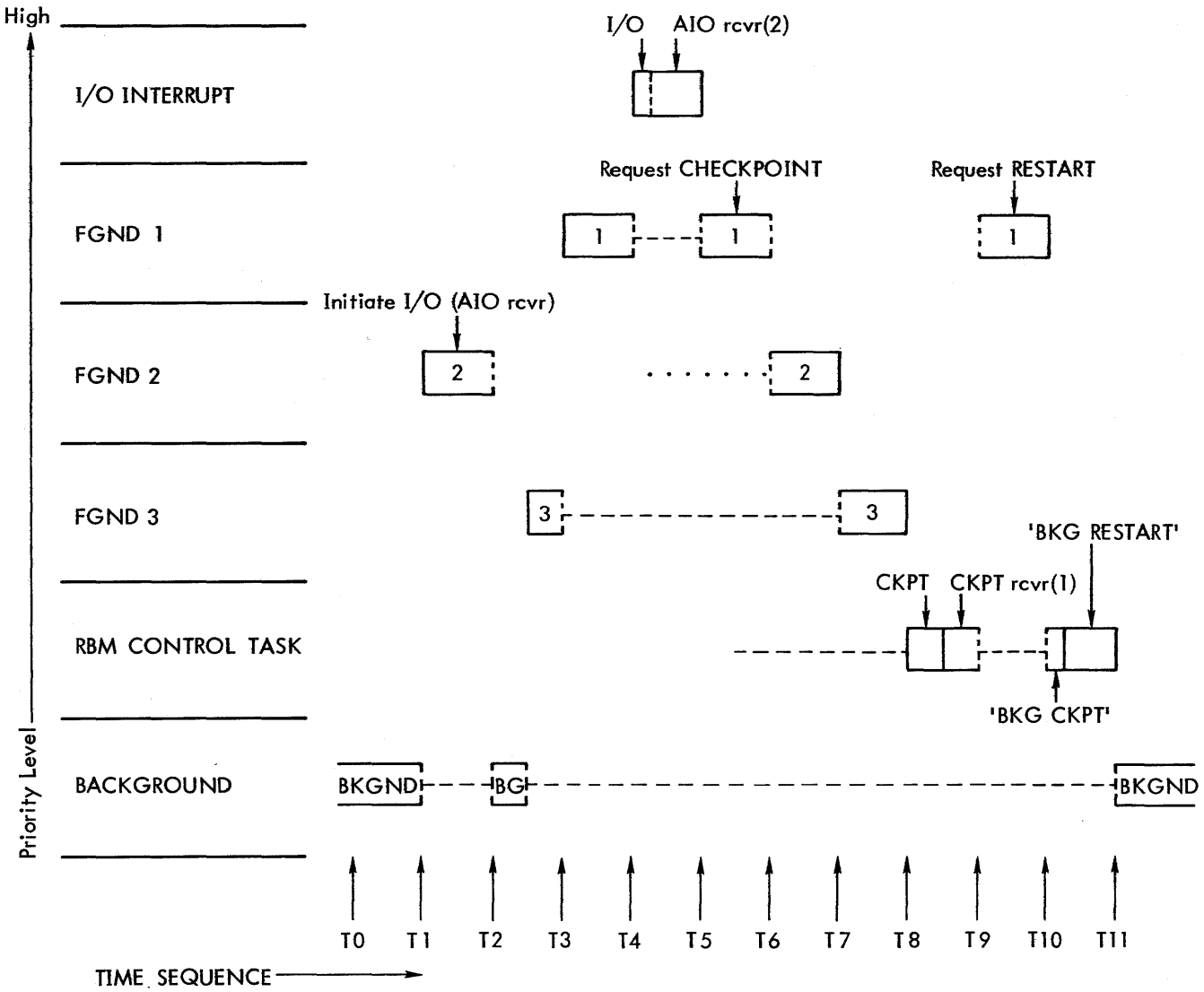
The design of the hardware priority system makes it unnecessary for the Monitor to be involved in the actual scheduling, and this procedure allows the tasks and programs to independently control the execution priority of certain operations within the foreground. For example, a real-time foreground task that is activated by an external interrupt may perform some processing and then issue a special Write Direct to trigger another related task to continue the processing at a higher or lower interrupt level. If the Write Direct is to a higher level, the interrupt to the higher level takes place immediately and the new task is begun. More frequently, the Write Direct is to a task at a lower priority level, and in this case the current task exits in a normal manner and the highest priority "waiting" task will become active. This task may or may not be the one that just received the Write Direct. Eventually, the task that received the Write Direct will be reached, and this task will then continue the processing at that level. Thus, real-time foreground programs can have an intricate scheduling scheme with no RBM intervention.

An example of interrupt-driven scheduling is illustrated in Figure 5.

LOADING FOREGROUND PROGRAMS

All programs must reside on the RAD in order to be read into memory for execution. They must be written onto the RAD by the Overlay Loader or the Absolute Loader. (See the !ABS control command description in Chapter 2 for restrictions regarding the use of the Absolute Loader.) In each of the methods described below, only the program root is loaded into memory from the RAD file as a result of the action taken. Segments must be read in by subsequent calls to M:SEGLD.

The most common method of loading a foreground program into memory is through a call to M:LOAD by another foreground program. The call takes place at the priority level of the foreground program and the request is placed into the queue stack. The program is actually loaded by the Monitor subroutine S:LOAD at the level of the RBM Control Task, and this method is the most logical one to be used. It is based upon conditions automatically detected by other foreground programs and requires no response or assistance from the operator.



Note: Times need not be equally spaced.

Time Point	Activity (Meaning)
T0	The background is executing.
T1	An interrupt is received for Foreground Task 2 which becomes active and saves the environment of the interrupted background task into its TCB.
T2	Foreground Task 2 requests an I/O operation, specifies an AIO Receiver, and exits. The background resumes processing.
T2.5	An interrupt is received for Foreground Task 3 which interrupts the BG.
T3	An interrupt is received for Foreground Task 1 which becomes active and saves the environment of the interrupted task (Task 3) into its TCB.
T4	At channel end, an I/O interrupt is received for the operation initiated by Foreground Task 2; the I/O Interrupt Task saves the environment of the interrupted task (Task 1). The AIO Receiver is entered at the I/O interrupt level and triggers Task 2, indicated by dotted line at FGND 2 level.

Figure 5. Foreground Priority Levels

Time Point	Activity (Meaning)
T5	The AIO Receiver returns via a RCPY L,P instruction. The I/O Interrupt Task exits, restoring the interrupted task's status. Foreground Task 1 resumes operation, requests a checkpoint of the background, and specifies a Checkpoint Complete Receiver. This action causes the RBM Control Task to be triggered, indicated by broken line at RBM Control Task level.
T6	Foreground Task 1 exits, restoring the interrupted task's status. This was actually Task 3, but Task 2 is waiting and it immediately becomes active.
T7	Foreground Task 2 exits, restoring the interrupted task's status. This was Task 3. It becomes active and continues from where it was suspended.
T8	Foreground Task 3 exits, restoring the interrupted taks's status. This was actually the background task. Since the RBM Control Task was triggered at T5, it is the highest waiting interrupt level. The RBM Control Task becomes active and stores the interrupted task's status into its TCB. The RBM Control Task calls the RBM subtask S:CKPT which writes the background into the RBM Checkpoint area on the RAD. S:CKPT then extends memory protection to the background and enters the specified Checkpoint Complete Receiver at the RBM Control Task level. In this illustration the Checkpoint Complete Receiver triggers Foreground Task 1 with a Write Direct instruction.
T9	Foreground Task 1 becomes active and saves the environment of the interrupted task in its TCB. The background area is now available to Foreground Task 1 for instructions and/or data. When processing is complete, Foreground Task 1 requests a restart.
T10	Foreground Task 1 exits, restoring the interrupted task's status (in the Checkpoint Receiver, which returns via a RCPY L,P instruction). The RBM subtask S:CKPT now completes its operation and returns to the RBM Control Task which calls in the subtask S:REST to restart the background task. S:REST first clears the background area, then reads the checkpointed background task in from the RAD. The background is then set "unprotected" which completes the restart operation.
T11	The RBM Control Task exits, restoring the status of the interrupted background task which then resumes processing.

Figure 5. Foreground Priority Levels (cont.)

Another method of loading a foreground program is through an unsolicited key-in by the operator. The operator must generate a Control Panel Interrupt and, in response to the request !KEYIN, type in "Q name", where "name" must be the name of a foreground program residing in the user processor area of the RAD. This action also results in a call to M:LOAD to queue the request. This method could be used in response to conditions detected outside the computer system (e.g., a certain time of day). Both of the above methods apply to semiresident as well as nonresident foreground programs. For resident foreground programs, they would be used only to obtain a fresh copy of a particular program without rebooting the entire system.

Loading through use of the queue stack requires use of the nonresident foreground area whether or not the request is for loading into this area. Therefore, whenever a nonresident foreground program is in core, all queue stack loading is suspended until the program occupying the nonresident foreground area releases the area by unloading.

Two other methods of loading foreground programs are available. They involve control commands that are normally used to load background programs that are part of a background job, and that must be preceded by an FG key-in. These commands are

!XEQ initiates loading from whatever RAD file to which background operational label OV is assigned. The method presumes that either the appropriate OV oplb assignment has been made, or that the program to be loaded is on the RAD file RBMOV to which the label OV is assigned by default.

!name causes the foreground program "name" to be loaded in the same way a background processor is loaded. The foreground program must reside in either the SP, FP, or UP area: they will be searched in that order. The user is responsible for avoiding duplication of program names in those areas.

The control command methods are closely tied to background schedules and do not provide adequate response to real-time needs. However, they can be used when debugging foreground programs.

LOADING NONRESIDENT FOREGROUND PROGRAMS

Nonresident foreground programs are also loaded by the Monitor service routine M:LOAD. Once loaded, these programs can be connected to an interrupt via an initialization routine or else can be triggered by a code given in the program's TCB. These programs then behave exactly like resident foreground programs. If the program just loaded resides in the area of core referred to as the nonresident foreground area, the nonresident foreground area is tied up until the program releases this space. A method is provided to automatically unload this area when M:ABORT or M:TERM is called by the task occupying the nonresident foreground area. Therefore, a FORTRAN program calls the library routine L:OP (generated by the compiler when the program calls STOP) to terminate and unload. If a FORTRAN program calls EXIT, the nonresident foreground area will not be unloaded.

FOREGROUND INITIALIZATION

When a foreground program is loaded, it may either be initialized by RBM (see Overlay Loader !$TCB card in Chapter 7) or may have its own initialization routine.

If the !$TCB card is used, the initialization routine will be entered at the interrupt level specified on the !$TCB card. The initialization code must therefore take the necessary precautions to ensure that it will only be executed once. It must then branch to the start of the program. When OLOAD builds the !$TCB, the task organization is

```
Orgin, from !$ROOT Card -  TEMP   START
                           :
TEMP SIZE, from !$ROOT     :
    Card                   TEMP   END

                           ADRL   PSD
                           :
See Table 18        TCB    :
                           TCB    16

                    PSD    DATA   0
                           DATA   0
                           STA    TCB + 10
                           RCPY   L, A
                           STA    TCB + 5
                           RCPYI  P, L
                           B      M:SAVE
                           ADRL   TCB
                           B      *$ + 1
                VECTOR     DATA   ENTRY POINT
First instruction of users code
```

If any module with an operand on the END card is loaded with the Root segment the last 'END' operand will be inserted into 'VECTOR' and will be entered when the associated interrupt level is triggered. Otherwise, "VECTOR" will point at the first word location cell of the Root.

In the case where the program contains its own TCB, the operand on the 'END' card will be entered at the level of RBM. The initialization code in this case must

1. Insert the PSD address into the dedicated interrupt location.
2. Arm, enable and perhaps trigger the associated interrupt level.
3. Perform any user-specified functions, such as special receiver connections, establishing foreground mail boxes, and so on.
4. Return to the 'L' register.

In this case, when the initialization routine executes at the RBM interrupt level, the RBM temp stack is available to the user code. This will allow enough temp for almost all monitor service calls.

If HEXDUMP was included at SYSGEN time, all Monitor service routines except M:RSVP may be used. If HEXDUMP was not included, the Monitor service routines M:RSVP and M:SEGLD may not be used.

For the benefit of segmented foreground programs, the initialize code (entered by M:LOAD) can assign an internal operational label to the foreground ML operational label. This internal operational label may then subsequently be used in calls to M:SEGLD. The foreground program may not use the ML operational label in calls to M:SEGLD.

If there is a loader built TCB, the initialization routine will be entered at the task level when its interrupt is triggered for the first time.

When foreground initialization is completed, the routine returns to RBM via RCPY L, P. Foreground initialization routines will also be executed any time the system is booted from the RAD if the task is flagged as a resident foreground task and resides on the SP, UP, or FP areas.

TASK CONTROL BLOCK FUNCTIONS

The Task Control Block (TCB) is a convenient means for organizing and storing information necessary to attain proper context switching, define dynamic blocking buffer pools, define temporary space necessary for reentrancy, and arm and enable the associated task. A foreground program may have one or more TCBs within the program (one for each task), but it is assumed that the first loadable item within a foreground program is a TCB. The TCB is used by the Monitor service routines M:SAVE, M:EXIT, M:LOAD, and by the Control Command Interpreter upon encountering a !C: command.

The TCB consists of 17 words and can be created at assembly time with Extended Symbol, or at load time by the Overlay

Loader. (A FORTRAN program must have its TCB created by the Overlay Loader). The TCB is usually a block of code contiguous to the task it describes, with address literals pointing to the temporary stack space. A DATA statement can set the initial code for the interrupt level state for the task interrupt level. The complete contents of the TCB are shown in Table 18.

Note: The code in TCB + 2 is the exact code used in the Write Direct that sets the interrupt level. This code is described in the appropriate computer reference manual.

Bit T in word TCB+1 indicates whether the task is using the Monitor I/O routines and the floating accumulator; if bit T is zero, a temporary stack is required and the M:SAVE routime will initialize locations 0001 through 0006, after saving the previous pointers for the interrupted task. If bit T is a 1 (meaning no floating accumulator and no temporary space are required), the M:SAVE routine will not set these locations. In a real-time environment it is recommended that a user does not set the T bit to 1 (the floating accumulator and temporary storage pointers are saved). The Monitor service routines M:SAVE and M:EXIT do not, themselves, use any temporary storage.

When the task is programmed in FORTRAN, the task entrance and exit, TCB, and task entrance procedure are set up by the Overlay Loader. The module load routine M:LOAD sets the pointer to the PSD into the dedicated interrupt location and arms, enables, and optionally triggers the associated interrupt level.

The background program will have a Task Control Block in protected foreground space.

Caution: Locations 1 through 5 in the zero table are not saved and are recreated from location 6. Thus, locations 1 through 5 must not be changed by a foreground program or they will not be the same after interrupt has taken place.

When the Overlay Loader creates the TCB for a foreground task, the items shown in Figure 6 are generated adjacent to the task. If the transfer address given in the object deck is relocatable 0, it is not treated as the entry point to an initialization routine, but is used as the entry address for that task. The task will be armed, enabled, and possibly triggered when loaded for execution depending on the contents of words 1 and 2 of the TCB, supplied to the Overlay Loader on the !$TCB card.

After a foreground program is loaded into core, certain items in the TCB are examined. A fatal load error results if the number of specified operational labels requiring blocking buffers exceeds the number of available blocking buffers (word 15 of TCB). If the number of available blocking buffers is sufficient, word 15 of the TCB is adjusted to reflect the current blocking buffer requirements.

Table 18. Task Control Block (TCB)

Location	Contents	Set by
TCB+0	ADRL PSD	Assembler/Loader
1	0 - 3: R-bit No. For WD; 4: T; 5: C; 6: X; 7 - 15: Dedicated Interrupt Location	Assembler/Loader
2	0: 0; 1: D; 2: 0; 3: 1; 4: 0; 5 - 7: Code; 8 - 11: 0000; 12 - 15: Int. Group No.	Assembler/Loader
3	ADRL TEMPBASE (temporary stack) (FWA)	Assembler/Loader
4	ADRL TEMPLIM (temporary stack) (LWA+1)	Assembler/Loader
5	Contents of L register from interrupted task	Current task (on actual entry)
6	Contents of T register from interrupted task	M:SAVE (or current task)
7	Contents of X register from interrupted task	M:SAVE (or current task)
8	Contents of B register from interrupted task	M:SAVE (or current task)
9	Contents of E register from interrupted task	M:SAVE (or current task)

Table 18. Task Control Block (TCB) (cont.)

Location	Contents	Set by
10	Contents of A register from interrupted task	Current task (on actual entry)
11	Contents of location 0006 (K:BASE) from interrupted task	M:SAVE
12	Contents of Location 0007 (K:TCB) from interrupted task.	M:SAVE
13	Dynamic base (K:DYN) for temp of current task; initially TEMPBASE + 6.	Assembler/Loader (changed by M:RES and M:POP)
14	Buffer pool LWA+1.	Assembler/Loader
15	Bits 11-15 contain number of buffers ($0 \leq n \leq 16$). Bits 0-7 are reserved for Monitor use and should be coded as zeros. Bit 8=1 indicates the first buffer blocked is reserved as a sharable buffer for packed files.	Assembler/Loader
16	"Use" bits for buffers in pool (0 if unused).	M:OPEN or M:CLOSE
PSD + 0	Interrupt task status flags.	Interrupt sequence
1	Interrupted task P register.	Interrupt sequence.
2	First instruction of current task. ⋮ Remainder of program (the PSD must be contiguous with the program but need not be contiguous with the TCB.)	Assembler/Loader

where

ADRL PSD is a pointer to the Program Status Doubleword. It is the location shown in the dedicated interrupt location when the interrupt takes place.

R-bit No. for WD is the hexadecimal value (from 0 to F) that indicates the register bit that identifies the particular interrupt level within the Interrupt Group (the hardware block of 16 possible interrupts).

T is the flag that indicates whether the M:SAVE and M:EXIT routines should set location 0001 to 0005; 0 means yes, 1 means no. (T must be 0 if any Monitor service routines are used.)

C is the flag that indicates whether the task is critical (see Glossary); 1 means yes, 0 means no. The default value is 0. This flag is provided for interpretation and use by the installation; RBM as distributed makes no distinctions based upon it.

X indicates whether or not the task is to be triggered at load time: 1 means yes, 0 means no. A code of 7 is issued subsequent to issuing the code (normally 2, "Arm and Enable") given in word 2.

D when set, indicates that no dismissal on wait I/O will be performed for this task.

Code is the interrupt system control code that indicates current or desired initial interrupt control status. The codes are 1 = disarm, 2 = arm and enable, 3 = arm and disable, 4 = enable, and 5 = disable, 7=trigger.

Buffer pool is an amount of space from 1 to 16 buffer areas in length, each of which is equal in size to the value contained in K:BLOCK.

"Use" bits are bits, from left to right, beginning with zero, showing which of the maximum number of buffers have been allocated by M:OPEN and have not yet been closed by M:CLOSE.

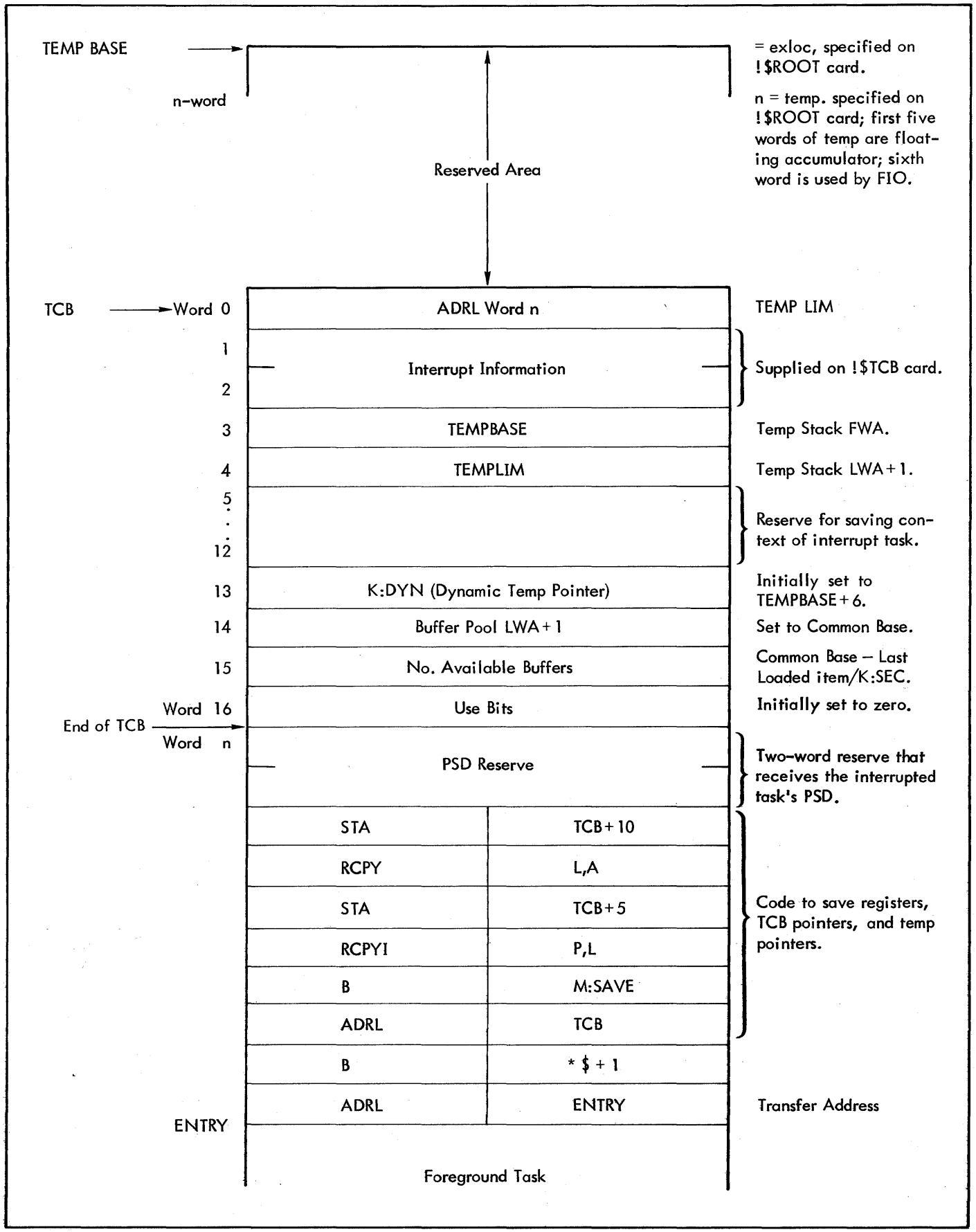


Figure 6. Task Entrance Format

In the event of a fatal load error in response to a load request from a background job stack via an !XEQ or !name command, the following message is printed on the DO:

```
!!BKGD XE ABORT LOCATION FFFF
```

If the request came from a queue stack load, the following message is logged on the DO:

```
NONRES FGND PGM xxxxxxxx LOAD ERROR
```

If a program has an initialization routine (that is, an end transfer address other than absolute or relocatable 0), that routine is responsible for storing word 0 of the TCB (the address to receive the interrupted task's PSD) into the dedicated interrupt location, as well as arming and enabling the appropriate interrupt level for each task within the program.

The initialization routine may also be used to assign any specific operational labels required by the program (e.g., the operational label or device unit number required) to read in subsequent segments.

If the program has no initialization routine, word 0 of the first loaded task (actually word 0 of that task's TCB) will be stored into the dedicated interrupt location for that task when the program is loaded. Next, the associated interrupt level is disarmed to remove any waiting interrupts; then it is armed, enabled, and possibly triggered, depending on the contents of words 1 and 2 of the TCB.

When a foreground task is activated, control is transferred to the address given in the dedicated interrupt location, where the interrupted task's PSD is stored, and execution resumes at PSD+2 at the level of that foreground program. This is a hardware function that preserves the interrupt status and execution location of the interrupted task. Next the register contents of the interrupted task must be saved.

Normally, the first instruction in a foreground program will store the contents of the accumulator into word 10 and the contents of the L register into word 5 of its TCB and then go to the Monitor service routine M:SAVE which will store the remaining register's contents into the active task's TCB. M:SAVE will also store the contents of K:TCB (used extensively by the Monitor to identify the currently active task) into word 12 of the TCB, and set K:TCB to point to the active task's TCB. If the active task requires temporary storage (word 1, T=0), the contents of K:BASE are stored into word 11 of the TCB and K:BASE is set to the first word address of the active task's temp stack. The floating accumulator is then set to point to the first six cells of the active task's temporary storage.

When the currently active task has completed all its operations, it exits through the Monitor service routine M:EXIT which restores the general register's contents and resets K:TCB and, if applicable, K:BASE. M:EXIT also performs a hardware exit sequence, by which it restores the interrupt status and the overflow and carry indicators, and returns to the interrupt task.

FOREGROUND PRIORITY LEVELS AND I/O PRIORITY

All foreground tasks that have a priority level lower than the I/O priority level and that operate without interrupts inhibited may use the Monitor I/O routines without any special restrictions. However, foreground tasks that have interrupts inhibited or have an interrupt level higher than the I/O priority level must not use Monitor I/O.

The recommended procedure for a task whose interrupt level is higher than the I/O priority level is to trigger a task whose priority is lower than the I/O priority. This lower priority task would then perform the required I/O operations. Generally, these high-level tasks are for emergency situations where no I/O is performed or when the task does its own I/O due to special requirements.

TASK DISMISSAL

When the SYSGEN option DISMISS is selected, the resident M:READ/M:WRITE code is extended so that any foreground task that elects to wait for on-going I/O to complete will be automatically suspended, allowing lower priority tasks (e.g., background) to proceed. This is accomplished by constructing an AIO receiver for the suspended task, which will reawaken the task when I/O completes. The implicit consequences of the scheme are:

1. If the task must forestall lower priority processing, it must be flagged for "no-dismissal" if I/O is performed (see description of TCB "D" bit, above). The RBM control task is flagged in this fashion.

2. Dismissal is transparent to the task; however, there is no overlapping of a task's I/O with computation while the task is dismissed. Overlapping of computation and I/O within a task can only be accomplished with "no-wait" M:READ/WRITE (and other) service calls; regardless of whether or not "DISMISS" is included.

3. Dismissal can occur on "no-wait" I/O but only when the requested device is already busy; in which case, the task will be suspended until the device becomes free and return will be made after the I/O is initiated. Double buffering can be achieved by a "no-wait" Read, followed by a "WAIT" write from a different buffer, followed by a "WAIT" check on the original read and then repeating the process. Such a task will proceed at the rate of the slower device.

4. By continually accessing only one device, a task may prevent lower priority tasks from accessing that device. Therefore, a foreground task that accesses the system RAD excessively may effectively suspend background.

AIO RECEIVERS

An AIO Receiver is a means whereby a foreground program can initiate an I/O operation, release control to lower level tasks, and regain control when the I/O operation is completed. The AIO Receiver itself is a closed subroutine which operates at channel end (or zero byte count, if specified) at the priority level of the I/O interrupt. It is used in conjunction with an I/O operation specifying "initiate only and return" (no wait). Typically, in order to maximize compute and I/O overlay, the foreground program will issue an I/O request with the "no wait" option and specify an AIO Receiver. When the I/O operation is successfully initiated, this foreground task exits from the active state (by a call to M:EXIT) and is restored to the active status at channel end by a Write Direct to trigger the interrupt level (from its AIO Receiver). The next I/O operation for that device file-number must be a "check" operation to complete the end-action of the file.

For I/O to RAD files, the AIO Receiver may be activated before the operation is actually complete. This will happen whenever a transfer across a disk track boundary occurs, more than X'1FFF' bytes are requested, or a flawed track is encountered. The calling task (not the AIO Receiver) must issue a "check" operation to complete the transfer. An AIO Receiver specified for the "check" operation will be honored.

Special considerations for use of AIO Receivers are:

1. The operation requesting an AIO Receiver is an "initiate and return" operation. If the device or the file is busy, the I/O operation is not initiated and a busy status is returned. It is the user's responsibility to determine the course of action to be taken at this point (e.g., loop until ready or ignore the operation).

2. If the file being used is a blocked file, an actual I/O operation may not be required, hence no channel end interrupt and no AIO Receiver operation. In this instance, the X register will be set to -1 to inform the user that the AIO Receiver will not be effective. A "check" operation is still required on the file before another I/O operation may be performed.

3. If a "check, no wait" is performed on a device that is busy with some file other than that specified by the check call, the check operation will be performed with an implied wait but only until the device is free for use by the specified file. For example, a busy status returned on a "check, no wait" operation always applies to the file specified by the Check call and if an AIO Receiver was specified, it will be honored.

4. If the AIO Receiver merely retriggers the task that initiated the operation, a danger exists in that it is quite possible for the AIO Receiver to operate before the task exits from its "active" state. Thus, the currently active task is retriggered, which results essentially in a no-operation. One means of avoiding this problem would be to have the AIO Receiver set a flag to inform the active task that it has run. In this way, the active task could inhibit interrupts prior to exiting, test whether the AIO Receiver has already operated, and if so, restore interrupt status and return to the start of the task. If examination reveals that the AIO Receiver has not run, the task merely exits through M:EXIT which will properly restore the interrupt status. Another means of avoiding this difficulty is to have the AIO Receiver trigger a task lower in priority than the active task. This lower priority task could retrigger the task initiating the I/O operation, thereby providing a positive trigger.

The form of the call to the AIO Receiver by the I/O Interrupt task is

```
LDA     aiodsb          (device status byte
                         from AIO in bits 0-7,
                         device number in
RCPYI   P,L              bits 8-12)

B       AIO Receiver Address
```

The AIO Receiver routine must return to the location contained in the L register on entry. All registers are assumed to be volatile, which means that they need not be saved and restored to their former contents. Because the AIO Receiver is processed at the priority level of the I/O Interrupt the processing in this routine should be of very short duration so as not to interfere with other I/O operations that may be in process. See also "End Action" in Chapter 5.

CLOCK1 RECEIVER

Extended zero table location X'1B4' contains a pointer to the CLOCK1 receiver chain. The S24RBM procedure file equates symbolic reference CLK1RXR to this location.

The receiver is entered at the counter 1=0 level. At entry, the A register will contain the actual counter 1=0 reentrancy count so that if it is desired to avoid repetitive operations where the counter 1 = 0 pulses have effectively stacked up, the receiver need only test for a change in the contents of the A register. All registers except A are considered volatile. The SYSGEN specification CLK1FREQ,n specifies the desired frequency (see SM Reference Manual 90 30 36) which defaults to 1/10 second but may be set to a value from 1/100 second to one second.

All receivers connect by first saving the current contents of the receive location CLK1RXR at their entry address -1 and then storing their entry address at CLK1RXR.

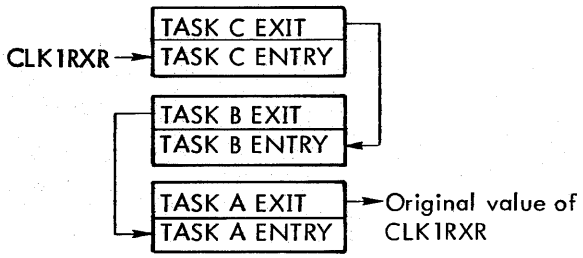
The delinking process requires a search of the receiver chain for the position within the chain of the delinking task and a substitution of the delinking's task exit address for that position within the chain.

Note that interrupts should be inhibited whenever the chain is manipulated. The following code might be utilized to connect and to delink from the chain.

To connect:

```
INHIBIT     R:PSW1
LDA         CLK1RXR
STA         MYENTRY-1
LDA         =MYENTRY
STA         CLK1RXR
RESTORE     R:PSW1
```

Assuming tasks A, B, and C had connected in that order to the CLOCK1 receiver, the CLOCK1 receiver chain would be as follows:



To disconnect:

```
            INHIBIT     R:PSW1
            LDX         =CLK1RXR
SEARCH      LDA         0,1
            CP          =MYENTRY
            BNC         $+2
            B           ITSME
            RCPY        A,X
            RADD        *Z,X
            B           SEARCH
ITSME       LDA         MYENTRY-1
            STA         0,1

            RESTORE     R:PSW1
```

CHECKPOINTING THE BACKGROUND

A foreground program may require use of the background area for either instructions or data. A checkpoint feature is included in RBM to allow access to the background area by a foreground program by writing any active background program onto the RAD and extending memory protection to the background area.

A checkpoint operation is initiated by a call to M:CKREST with the appropriate option. M:CKREST will return a status specifying whether or not the request was honored. The request will not be honored if the background has already been either checkpointed by a foreground request or automatically checkpointed as a result of loading a nonresident foreground program extending into the background. It is the responsibility of the user to schedule the use of the background space by foreground programs. The actual checkpointing is accomplished either at the priority level of the RBM Control Task or at the priority of the calling task.

If the checkpoint is performed at the priority level of the calling task, a return from M:CKREST with a status of zero (A = 0) indicates that the checkpoint has been performed. If the checkpoint is to be performed at the level of the RBM Control Task, the requesting program must exit its "active" state to allow the checkpoint operation to be performed. The program requesting the checkpoint would generally specify a "Checkpoint Complete Receiver". This receiver is operated at the priority level of the RBM Control Task when the checkpoint is complete.

The receiver will generally retrigger the requesting program to inform it of the completion of the checkpoint. Return from the Checkpoint Complete Receiver is to the location contained in the L registers on entry. All registers are assumed to be volatile, and need not be saved and restored to their former contents.

When the foreground program no longer requires use of the background area, it should restart the background task by a call to M:CKREST with the "restart" option.

7. OVERLAY LOADERS

Two loaders, OLOAD and BLOAD, are provided with RBM. Functionally, they are quite similar. The two major differences are (1) Public Library loading is supported only by OLOAD, and (2) BLOAD creates a load module one granule at a time. Thus, OLOAD runs faster than BLOAD, but BLOAD can load program segments larger than the available loading space. In this chapter, statements or paragraphs applicable only to OLOAD or to BLOAD are indicated parenthetically.

The Overlay Loaders can be used to create overlay program load modules for later execution in either the foreground or background. Overlaid programs can be permanently entered (as a file) into either the system or user processor areas, or into a temporary overlay file. Since they are stored on the RAD as an absolute core image, they can be quickly loaded into memory for execution.

A general overlay structure is illustrated in Figure 7. The structure is restricted to a permanently resident root segment and up to 255 overlay segments. (For background and nonresident foreground programs, the permanent root segment is resident only during actual execution.) For foreground programs, the TCB and the initialization routine (if one is present) must be in the root segment, but data and instructions can be located in both the root and the overlay segments.

A Blank COMMON data area can also be established for use by the root and overlay segments.

Each segment is created by the Overlay Loaders from one or more object modules (assembly language, FORTRAN, or RPG output). The control commands required to create the overlay segments are defined in this chapter. During execution, the Monitor service routine M:SEGLD is used to control both the loading and the transfer of control between various segments.

The overlay segments must be explicitly defined at load time and explicitly called at execution time. There is no provision for automatically calling in a new overlay segment by a subroutine reference. However, the subroutines on a particular path may communicate with each other, with the restriction that it is the program's explicit responsibility to ensure that any subroutine referenced is currently in core.

The Overlay Loaders accept input in Xerox Standard Object Language from predefined, prepositioned files, and prepare output in absolute core-image form on the RAD to be read by the RBM Loader (M:LOAD) for later execution in either foreground or background areas. If a resident or nonresident program can tolerate a loading delay of 20 to 100 milliseconds, foreground or background programs of virtually unlimited size can be constructed by the use of overlays despite limitations in available core storage.

In creating core images on the RAD, the Overlay Loaders perform the following functions according to user options:

- Satisfy external reference/definition linkages and resolve forward reference and displacement chains.
- Search specified libraries for unresolved references and load these selected routines into core memory.
- Build the OV:LOAD table for the loading of overlay segments.
- Write the overlay cluster onto the OV file.
- Allocate COMMON.
- Allocate temporary storage stacks.
- Create a Task Control Block (TCB) and initialization information.
- Create the Public Library and associated transfer vectors (TVECT)(OLOAD only).
- Output maps of segment names and addresses, external definitions, and information concerning COMMON and temporary areas to the LO device.
- Allocate, initialize, and satisfy reference linkage for Labeled COMMON.

OVERLAY CLUSTER ORGANIZATION

The overlay cluster is the collection of absolute overlays formed by the Overlay Loaders from relocatable binary object modules. (Note that the Loaders do not accept an absolute load origin in any input module.) An overlay cluster usually consists of two principal sections: the root segment and the overlay segments although it may consist of only a root segment. Each segment consists of one or more binary modules and associated library routines. Overlay segments are numbered in any order by the user, except for the root segment, which is always designated as segment 0. Those segments in core memory at any one time form a path. An overlay cluster with several paths is shown in Figure 8. Segments are shown as horizontal lines and, in this example, are numbered in the order in which they are built by the Overlay Loaders. Note that a given node, each path associated with a branch must be completed before a new branch is connected to this node.

The overlay cluster shown in Figure 8 consists of a root and segments 1 through 15. Segments 0, 1, 3, 4, 5, 6 constitute a path. On the RAD or disk pack the root is preceded by a file header, one RAD granule in length, that contains information by which the RBM Loader M:LOAD can correctly read the root. The root is resident at all times during execution of the overlay program and contains information (OV:LOAD table) for loading of the remaining overlay segments.

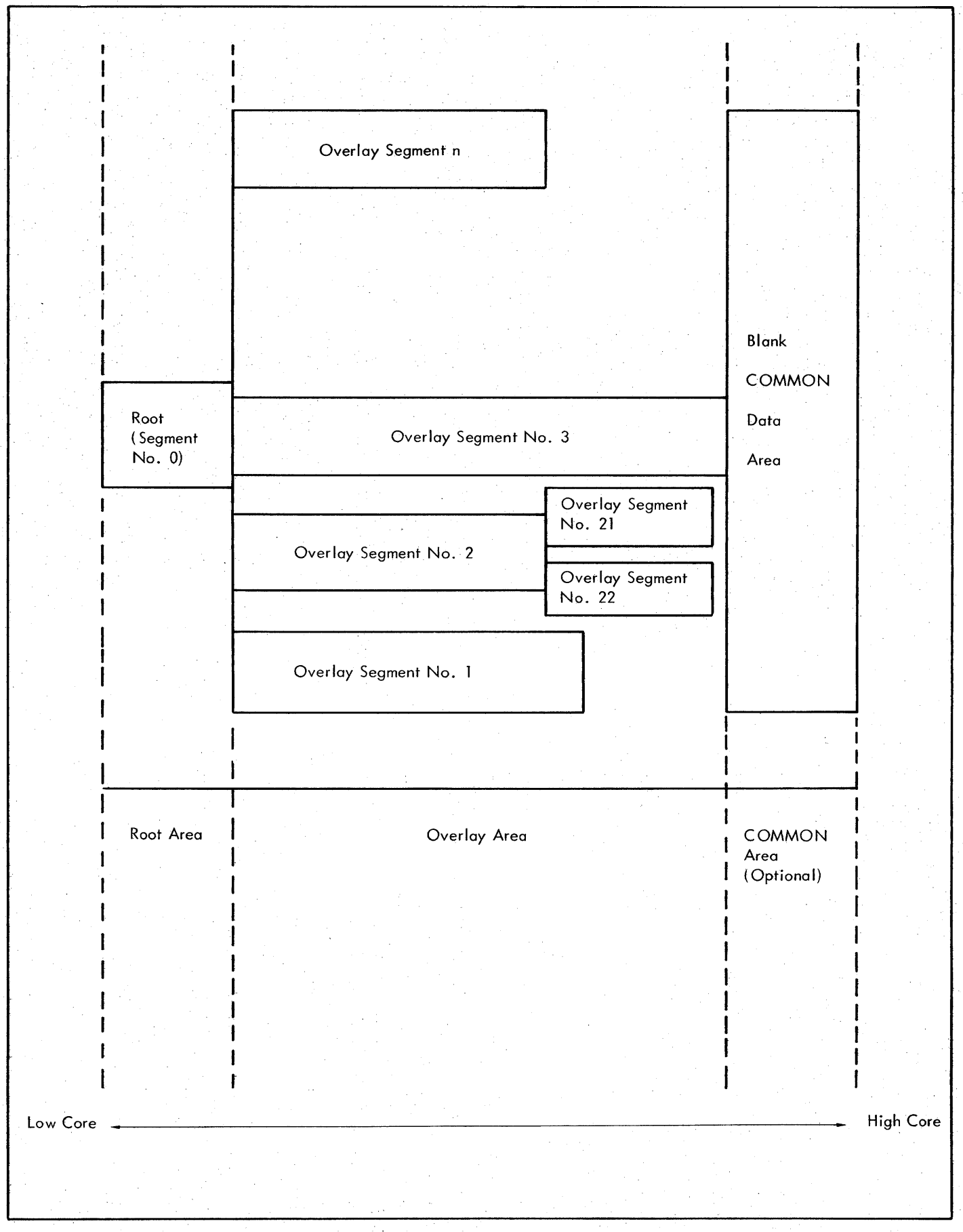


Figure 7. General Overlay Structure Example

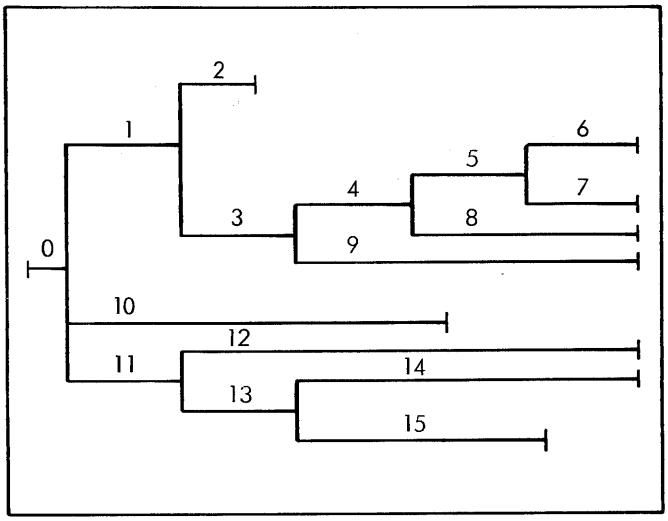


Figure 8. Sample Overlay Cluster Configuration

When first defined along a path by an object module, Labeled COMMON will be allocated preceding that module. Should the same Labeled COMMON be subsequently defined by another module, the area prescribed should be no greater than that already allocated, and reference to the initial definition will be provided. Allocated space for Labeled COMMON is cleared to zero entries except where data is provided by modules of the same segment (or root). An XSYMBOL subroutine may access Labeled COMMON via an external reference (REF or SREF) if the Labeled COMMON is defined in a previously loaded module.

Library modules of the root may not initialize Labeled COMMON allocated in the program portion of the root. The number of Labeled COMMON blocks associated with a module is limited to 40.

Communication between segments by external reference/definition linkages is subject to the following restrictions:

1. No segment in a path may reference a segment in another path.
2. The user must ensure that all communicating segments are in core memory during execution.
3. Because the Overlay Loaders will satisfy a linkage only within a path, identical references and definitions may be used in different paths that do not contain a common segment. However, the user must avoid references to the same definition in different higher level segments.
4. Library search procedures for a User or System Library restrict the use of unique library DEFs and REFs to a maximum of 300 along any path of the program.
5. Forward references in library modules of the root are disallowed, and it is suggested for good programming practice that User Library programs not make references outside the library realm.

To satisfy any remaining unsatisfied primary references, the Overlay Loaders search the following libraries in the specified sequence:

1. Public Library
2. Monitor Service Routines
3. Basic or Extended Library (optionally)
4. Main Library

CORE LAYOUT DURING LOADING

Background memory during the operation of the Loader is divided into four areas:

1. A fixed area large enough to contain the background temp stack, the Loader root, and the Loader overlays.
2. The segment table, fixed at 10(n+1) where n equals the number of segments.
3. In OLOAD, a dynamic area in which the segment is loaded. In BLOAD, a fixed (granule sized) length block for segment loading.
4. A dynamic area containing the symbol tables (allocation is five to eight words per symbol).

If areas 3 and 4 overlap at any point in the load process, overflow occurs and loading aborts.

OVERLAY LOADER OPERATIONAL LABELS

The Overlay Loaders reference the operational labels listed below. Some assignments are user-defined, while others are handled internally by the Job Control Processor or by the Loader itself. All other operational labels referred to an !$LD cards must be assigned and positioned by the user prior to the !OLOAD or !BLOAD command.

Label	Explanation
CC	Control commands. If a KP key-in is in effect, control commands are read from OC.
DO	Diagnostic messages. The default assignment is that given by the Job Control Processor on reading a !JOB card.
GO	Sequential-access file that contains object modules to be processed by the Overlay Loaders. Object modules are written onto GO by a preceding processor. The Loaders rewind GO initially, and also after loading is completed. GO receives a default assignment by the Job Control Processor to the permanent file RBMGO in the System Data area.

Label	Explanation
LO	Maps.
LI	Assigned internally for file I/O.
LL	Log of control commands.
OC	Abort messages and Overlay Loader messages that require operator attention. Control commands are read from OC if a KP key-in is in effect.
OV	Output file (random format) for the Overlay Loaders containing the completed overlay cluster. If the user wishes to have the overlay cluster in a permanent file, he must key in SY (for write-protected files) and assign OV to that permanent file. By default, OV is assigned to the permanent file RBMOV in the System Data area.
PI	Used for loading the Overlay Loaders' own overlays. PI is assigned by the Job Control Processor.
XI	Temporary RAD or disk pack scratch file containing the symbol table for each segment. XI is assigned by the Job Control Processor.
ID	An optional operational label used to write the idents of nonlibrary programs for use by Debug at execution time. If the user assigns ID, the assignment must be for a packed file that has a record length of five words. By default, ID is assigned by the Job Control Processor to RBMID (a one-sector file) in the System Data area.

MAP

Three types of maps may be output to the LO device following PASS2, according to one of three Map control commands that may be input: a Short Map (!$MS), Long Map (!$ML), or Program Map (!$MP). If no Map control command is specified, no map will be output.

Figure 9 shows the format for a Long Map. Note that DEFs in the Permanent Symbol Table are mapped after the Overlay Task line. The format for a Program Map would be the same as the Long Map except that library and Permanent Symbol Table symbols are suppressed. The lines of the map that are flagged with an asterisk (*) show the format and output of a Short Map (in an actual Short Map no asterisk would appear in the listing). A definition of each item of the map is included in Figure 9.

```
*MAP

*OVERLAY TASK {FO|BA} ORG = xxxx  HLLOC = xxxx  CBASE = xxxx  CSIZE = xxxx  UMEM = xxxx  SECT = xxxx

*ROOT  ORG = xxxx  LWA = xxxx  LEN = xxxx  TRA = {NONE|xxxx}  SEV = xxxx  OV:LOAD = xxxx

       [f1]  DEF   symbol  {L|I}{S|U|P}{B|E|M}  yyyy  DEF symbol...etc.

       [f1f2]REF   symbol  {L|I}{S|U|P}{B|E|M}  zzzz  REF symbol...etc.

*SEGMENT     IDENT    NODE    ORG     LWA     LFN     TRA     SEV
             xxxx     xxxx    xxxx    xxxx    xxxx    xxxx    xxxx

       [f1]  DEF  etc.
             .
             .
             .
       [f1f2]REF  etc.
             .
             .
             .
             REF


*SEGMENT
 .
 .
 .
*SEGMENT
 .
 .
 .
*ERRSEV = xxxx

*END MAP
```

Figure 9. Long (Load) Map Format

where header keywords have the following meaning:

Overlay Task Keywords

ORG	First word address of the Overlay Task area. It is the FWA of the Temp stack.
HLLOC	Last word address of longest segment.
CBASE	Base of Blank COMMON.
CSIZE	Largest Blank COMMON size encountered.
UMEM	The number of locations between the end of the longest path, and either the beginning of Blank COMMON or the end of the assigned task area.
SECT	The number of sectors required to store entire overlay cluster.

Root Keywords

ORG	FWA address of the root. In the foreground, this is assumed to be the address of the TCB; in the background, it is the FWA of the root.
LWA	Last word address of the root segment. The area from ORG to LWA includes the root code and the OV:LOAD table (and in the foreground, the TCB).
LEN	LWA-ORG+1.
TRA	Background – last end transfer encountered on a module used to form the root. If there is no transfer address, 'NONE' is output. Foreground – the entry address of an initialization routine that arms and optionally triggers interrupts at run time. If the Loader builds the TCB, it is assumed that no such initialization exists and TRA=NONE.
SEV	Error severity encountered during loading binary modules. Taken from the END item of the binary module.
OV:LOAD	Address of the OV:LOAD table.

General Keywords

f_1f_2	Error and identifier flags preceding external definitions and references. Possible flags are: D Double definition or reference. LC Labeled COMMON U (DEF) – a definition declared, but given no value. U (REF) – reference unsatisfied in this path. P Primary reference. S Secondary reference.
DEF	An external definition.
REF	An external primary or secondary reference.
symbol	DEF/REF name of one to eight EBCDIC characters.

Figure 9. Long (Load) Map Format (cont.)

General Keywords (cont.)

L/I	Library or Input REF/DEF.
S/U/P	System, User, or Public Library.
B/E/M	Basic, Extended, or Main mode.
yyyy	Value of a DEF.
zzzz	The number of the segment in which this reference was satisifed. For unsatisfied references, zzzz is blank.

Segment Keywords

IDENT	Numerical identifier of this segment as found as the first parameter on the !$SEG card.
NODE	The numerical identifier of the segment to which this one will be attached. If NODE is the root, 0 is output.
ORG	Beginning location (execution) of this segment. The point in core at which loading begins. The first reserves before data in a segment are not output.
LWA	LWA of this segment. Includes areas defined by RES and ORG.
LEN	LWA-ORG+1.
TRA	The last encountered transfer address is placed as an entry point in the OV:LOAD table for this segment.
SEV	Same as for ROOT.
ERRSEV	Total error severity for loading process. If any SEV > 0 or there are unsatisfied primary references, ERRSEV=1. Only in forming a PUBLIB do double DEFs or unsatisfied secondary references cause ERRSEV=1. Errors in the input binary may cause ERRSEV=2.
END MAP	Completion of loading process.

Figure 9. Long (Load) Map Format (cont.)

Certain reserved DEFs will be output by the Loaders. These are:

P:FWA	Program First Word Address
P:LWA	Program Last Word Address
P:TCB	Primary TCB FWA (if the Loader builds the TCB, otherwise, not generated)
P:RLWA	Root Last Word Address is an overlaid program (suppressed for root-only programs)

These are treated as external definitions and may be referenced by the program.

P:LWA and P:RLWA are restricted to definition by the Loader. User definition of these symbols will result in indeterminate results. They may, of course be referenced by user code.

P:FWA and P:TCB may be user defined, but they will be flagged as duplicates and otherwise ignored.

CALLING OVERLAY LOADER

The Overlay Loaders are requested via an !OLOAD or !BLOAD command which causes the root segment of the Loader to be read into core memory from the RAD. The form of the command is

!OLOAD [segments, {F / B} ,S,D,X,cmn][,R]

or

!BLOAD [segments, {F / B} ,S,D,X,cmn][,R]

where

segments denotes the number of segments in the overlay cluster. If "segments" is not specified, a zero is used, denoting that only a root segment is to be loaded. The value of the segments parameter may exceed the actual number of segments to be loaded.

F or B specifies either a foreground (F) task or a background (B) task. The default case is background.

S specifies a step mode of loading to be used for paper tape input.

D indicates the ident of each nonlibrary module is to be written to operational label ID for use by Debug at execution time.

X indicates that the Loader is to abort the job if a severity error greater than zero is encountered during loading. The loading procedure is completed and the map is output.

cmn for background tasks, cmn denotes an optional Blank COMMON size. For foreground tasks, cmn denotes a base for Blank COMMON. A check is made at the end of the load to determine whether the Blank COMMON allotment overlaps the root. If it does, the warning message $$ ERR CO is printed but no error severity level is set.

R for foreground tasks only, specifying this parameter causes only the root size to be entered into a sector header (OV:LOAD table) instead of the program's longest path.

This action is intended for the use of a foreground program that only occasionally uses a large data buffer. A program of this nature can reside in foreground without checkpointing background until such time as it requires background space. Caution must be exercised in the use of this parameter, since the background must be explicitly checkpointed and restored, when necessary, by the foreground task.

When the step mode of loading is defined, the operator is notified after the loading of each module from paper tape by the message

```
!!BEGIN WAIT
```

Depressing the console interrupt button and keying in an S will initiate either the loading of the next module from the paper tape unit or the reading of the next control command. An X response causes the loading process to abort.

In allocating COMMON for background programs, the Overlay Loader compares the cmn parameter with the first nonzero COMMON size allocation value encountered in loading and employs the larger of these two values. The COMMON base is set by subtracting the COMMON size from K:UNAVBG.

BLANK COMMON ALLOCATION IN FOREGROUND LOADING

For foreground loads, the Overlay Loader allocates Blank COMMON and blocking buffer pools in accordance with the rules delineated in Table 19.

Reading an !EOD control command causes the Overlay Loader to satisfy forward references, output any specified map, close files, and return control to RBM via M:TERM. The form of the command is

```
!EOD
```

CONTROL COMMAND FORMAT

Except for the !OLOAD and !BLOAD commands which are read by the Job Control Processor, the Overlay Loader control commands are read from the CC device under Loader control, unless a KP key-in is in effect, in which case control commands are read from the OC device. The general format of control commands is

```
!$mnemonic parameter
```

where

! identifies the record as a control command.

$ indicates that the control command is unique to the Overlay Loader.

mnemonic is the code name of an Overlay Loader control command and begins immediately following the !$ characters.

parameter is a series of optional or required parameters unique to the specific command. The formats of parameters are (1) a decimal integer of up to five positive numbers but having a value less than 32,767; (2) a hexadecimal string of the form ±xxxx; (3) an EBCDIC string of up to eight characters but not exclusively characters 0 through 9; or (4) a string of the form EBCDIC string ± hexadecimal number.

From one through eight blanks are permitted between the mnemonic and the first parameter. If more than eight blanks are detected, the parameter list is considered empty.

The only allowed delimiter between parameter fields is a comma; no embedded blanks are allowed in or between any fields. A single blank terminates the parameter string. Two successive commas indicate an empty field. Comments are allowed on a control card.

CONTROL COMMAND REPERTOIRE

BLOCK The !$BLOCK control command will allocate blocking buffers from unused memory space as requested

Table 19. Foreground Load Blank COMMON Allocation

Program Type	cmn Specification	CBASE	Program Limit	BB Pool End
Resident Foreground	< Program origin	cmn	FGLWA	!$BUFEND (required)
	> Program origin	cmn	cmn	!$BUFEND (required)
	Not specified	FGLWA-CSIZE	CBASE	!$BUFEND (defaults to CBASE)
Nonresident Foreground	< Program origin	cmn	BGLWA	!$BUFEND (required)
	> Program origin	cmn	cmn	!$BUFEND (required)
	Not specified	FGLWA-CSIZE	CBASE	!$BUFEND (defaults to CBASE)
Nonresident Foreground with R Option	< Program origin	cmn	BGLWA[tt]	!$BUFEND (required)
	> Program origin	cmn	cmn	!$BUFEND (required)
	Not specified	BGLWA[t]	CBASE	!$BUFEND (required)

where

CBASE is the first word address of COMMON.

CSIZE is the size of the first COMMON declaration encountered.

FGLWA is the last word address of foreground (K:BACKP-1).

BB POOL END is the blocking buffer pool last word address plus 1.

BGLWA is the last word address of background.

cmn is the COMMON specification parameter on the !OLOAD command.

[t] If Blank COMMON is encountered in the root, a warning is issued ($$ ERR C1), CBASE is set to FGLWA-CSIZE and the R option is ignored.

[tt] If the root exceeds FGLWA, a warning is issued and automatic checkpoint will occur at program core-load time.

either by a count or by defining operational labels that may require blocking buffers at run time. The list of such labels along with limits of available memory will be passed via the file header to M:LOAD, which will allocate a blocking buffer pool at run time. The pool will be utilized dynamically to provide blocking buffers in cases where a call to RBM routines M:READ or M:WRITE is not preceded by a call to M:OPEN. A call to M:CLOSE may release any such buffers. Thus, if two operational labels were to use a blocking buffer area at different times, the first might release the area for use by the second. Only one of the two labels would be required on the !$BLOCK command.

M:LOAD checks which of the operational labels are assigned to block files at run time to make the pool allocation. If such an allocation overflows the available memory space (between the end of the longest path and COMMON), the execution aborts. However, the user may define his own blocking buffer by specific calls to M:OPEN. Such an area should be in a reserved area of his own path. He should not use the dynamically allocated pool area, and blocking buffers may not be allocated in temporary stacks.

Only one !$BLOCK command is allowed in a single job step, except when used with multiple !$TCB commands. The format of the !$BLOCK command is

$$!\$BLOCK\left\{\begin{matrix}oplb_1[,oplb_2,\ldots,oplb_n]\\ ALL\\ c\end{matrix}\right\}[,S]$$

where

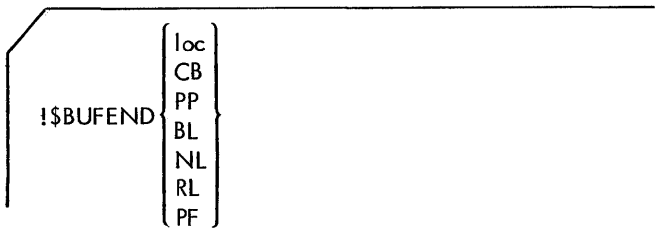
$oplb_i$ defines an operational label (which is a two-letter mnemonic or a FORTRAN device unit number; e.g., BI, SI, F:106. The $oplb_i$ parameter may not be a device-file number or file name. The oplb must be assigned to a blocked file. Only 10 operational labels will be read; additional ones will be ignored. In lieu of operational labels, the user may provide a count (c) of blocking buffers required.

ALL results in the entire area of unused memory (UMEN) being used as blocking buffers to a maximum of 16.

c defines a count of buffers required to a maximum of 16.

S indicates that the first of the indicated buffers is to be set aside as sharable by packed random files (see "Blocking Buffers", Chapter 5).

BUFEND The !$BUFEND command must be used to specify the LWA+1 of the blocking buffer pool for foreground loads if required by the rules specified in Table 19. Only one !$BUFEND command applies during a load sequence. Buffer requirements must be specified by an !$BLOCK command. The lack of buffer specification, or overlap with COMMON, program, Public Library, or Monitor areas will cause an Overlay Loader error message ($$ERR BU). The format of the command is



where

loc is a decimal or hexadecimal address.

CB indicates that pool LWA+1 = CBASE.

PP indicates that pool LWA+1 = program LWA plus pool size.

BL indicates that pool LWA+1 = BGLWA.

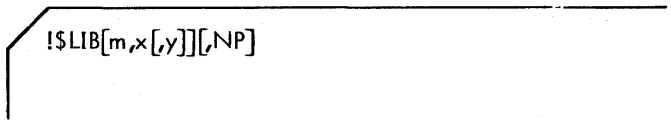
NL indicates that pool LWA+1 = nonresident FG LWA.

RL indicates that pool LWA+1 = resident FG LWA.

PF indicates that pool LWA+1 = program FWA.

See also "COMMON Allocation for Foreground loading."

LIB The !$LIB control command specifies the library search sequence for the entire load process, or from that point in the OLOAD control command sequence at which it occurs. If the !$LIB command is not present, OLOAD follows the default case (Basic System Library search). The format of the command is



where

m specifies the search mode and is one of the following EBCDIC codes:

Code	Search Mode
B	Basic (and Main)
E	Extended (and Main)
M	Main only

x[,y] specify the order of search, and are either of the following EBCDIC codes:

Code	Library
S	System
U	User

The order in which x and y are specified determines the order of library search. If only x is specified, y will not be searched.

NP specifies suppression of Public Library linkage if the !$LIB command precedes a !$ROOT command. If the NP parameter occurs on a LIB command following a !$ROOT command, or in a PUBLIB load, NP is ignored; any other parameters in the command are interpreted as described, however.

An !$LIB command with no parameters or with only the NP parameter will suppress nonresident library search from the point of its occurrence in the OLOAD control command stream.

MS,ML,MP The Map control commands specify that map information is to be output on LO. The three forms of map commands are shown below.

If the !$MS (Short Map) control command is specified, only root and segment headers will be output. Also output is a summary containing the origin of the overlay program, the length of the longest path, temp stack size, memory that is available for the blocking buffer pool, and the COMMON base. The format of the command is

If the !$ML (Long Map) control command is specified, the short map plus external references and all external definitions and their values including the libraries and permanent symbol table are output. Double definitions, and definition declarations that were not given a value are flagged D and U, respectively. Unsatisfied primary references are flagged with UP, unsatisfied secondary references with US. The format of the command is

```
!$ML
```

The output of the !$MP control command is identical to that of !$ML, except that library definitions and references and the permanent symbol table are suppressed. The format of the command is

```
!$MP
```

If relevant, information concerning the Public Library is also mapped.

TCB The !$TCB control command indicates (for a foreground task only) that the Overlay Loader must create a TCB and reserve a PSD location, and must generate a call to RBM routine M:SAVE. M:FSAVE will be called if the set multiple precision mode exists. In addition, information to initialize the TCB at run time will be passed in the file header. If no !$TCB command is present, it is assumed that a TCB has been assembled into the root segment. Since the background TCB lies in protected memory, it cannot be assembled into the root of the background overlay cluster, but the necessary information is passed by the Loader to M:LOAD via the file header. Therefore, the TCB option applies to foreground tasks only. Multiple !$TCB commands may be used internal to the root program. Each !$TCB command would connect a separate interrupt function to the root program and be followed by !$LD commands to load associated modules. The !$TCB may be followed by a !$BLOCK command that would identify independent buffer blocks with its function. Individual temp stacks will be reserved by other than the initial !$TCB command that must precede the !$ROOT command. The format of the command is

```
!$TCB w1,w2[,temp]
```

where

w_1, w_2 are the values to be placed in words 1 and 2 of the created TCB (see "Real-Time Programming," Chapter 6).

temp defines the size of the temporary stack to be reserved for a TCB other than the initial TCB.

The Overlay Loaders will handle specific and default cases of program execution and TCB initialization within the framework of the following restrictions:

- The Overlay Loaders define all background Task Control Blocks completely, using the value of the temp parameter on the !$ROOT card, load information, and the !$BLOCK parameters.
- In foreground tasks, if the user assembles the TCB as part of the program, it either must contain all information as data or as external references satisfiable at load time, or be initialized by the task itself. A transfer address is assumed to be a transfer to an initialization section that will do any required housekeeping, arming, enabling, or triggering the task. If no transfer address exists, M:LOAD will arm and enable and, optionally, trigger the task using information in words 1 and 2 of the TCB.
- If the Overlay Loaders initializes the TCB by means of the TCB parameters, they do so completely, using load information and values on the !$TCB and !$BLOCK cards. No partial initialization of a TCB is allowed with the exception of the blocking buffer pool. If a user builds his own TCB, the TCB must begin at the execution location plus the "temp" value specified on the !$ROOT command.
- For foreground tasks for which the Loader builds a TCB, the Loader will create the PSD reserve and a call to M:SAVE. The user's root is then entered either at the location specified in the transfer address, or at the FWA of the root when the transfer address is missing. The map will indicate a transfer address of "NONE" for the root.
- Where multiple !$TCB commands are used within the root program, the transfer address for the program is established by the modules preceding a second use of the !$TCB. FORTRAN generated programs do not provide a transfer address. If no transfer address exists, each subtask within the root program will be initialized by M:LOAD using the information in words 1 and 2 of their respective TCB. If a transfer address is provided, M:LOAD will not initialize any subtask.

The user exits with either a call to the RBM routine M:EXIT or by a standard exit procedure.

Public Library routines and Monitor service routines called by the user program will require temporary storage areas that are dynamically allocated at execution time. These temporary storage areas must be allocated in a fixed storage stack that is reserved by the Loader at load time on the basis of the temp parameter on the !$ROOT control command. In addition, the Loader will insert in the TCB the first and last word addresses of the area. The temp area will be allocated preceding the root segment. It need not be a reserve in the module.

For more information on initialization and structure of TCBs, see Chapter 6.

ROOT The !$ROOT command specifies that the modules that follow it constitute the root segment of the overlay cluster. A !$ROOT command must precede all !$SEG commands, and may be followed by !$LD, !$INCLUDE, !$EXCLUDE, !$TCB, !$LCOM, !$RES, !$MD, !$LIB, and !$LB commands, which cause the loading of those modules that form the root segment. Loading of the root will begin at the first cell following the temp stack for the background task. An execution bias may be specified. The user must ensure that the root segment, exclusive of any library loading, is less than 32K bytes. The root and its library are written as two records. Therefore, the library portion of the root may also be a maximum of 32K-1 bytes, which gives a maximum root size of approximately 32K words. The format of the command is

```
!$ROOT[temp,exloc,oplb,n][,DT]
```

where

temp defines the size of the overlay cluster's temporary stack needed for the largest possible nesting of Public Library and Monitor service routines. The default size is 80 cells. If a TCB has been assembled into a foreground program, zero should be used for temp.

exloc specifies the beginning location of the area in memory that the overlay cluster will occupy at execution time. The default case is K:BACKBG for a background task and K:NFFWA for a foreground task. The temp stack will be allocated at exloc.

oplb,n specifies that n modules are to be loaded contiguously from the operational label oplb.

DT specifies that calls to M:PUSH, M:PUSHK, M:PUSHC, M:PSHC, and M:RES are modified to dynamic-temp storage (in the calling sequence, "ADRL temp" is changed to "DATA 0", and trailing reserve is stripped). This is done only for those ROMs (including library modules) loaded by this command.

Note that if the oplb parameter is absent, !$LD (Load) or !$INCLUDE control commands must follow !$ROOT to specify loading. If oplb is present but the n parameter is not, loading proceeds from oplb until an EOF status is encountered.

LD The !$LD control command identifies one or more modules to be loaded as part of a segment. Each input file must be ordered in the same sequence as the !$LD cards in the control stack accessing that file. The Overlay Loader reads only relocatable binary modules from the GO file and other input files specified on !$LD, !$SEG, and !$ROOT cards. All files must be pre-positioned (GO is rewound by the Loader), and the modules must be in the same position on each file as calls on that file. The use of the IDNT on the !$LD card ensures the loading of the proper module. Note that the file must be positioned to the proper module in the file when the Loader reads from that file. Since there are no file-positioning control commands recognized by the Overlay Loader, each file must be constructed in correct sequential order. The form of the command is

```
!$LD [oplb][,{ident}][,DT]
             {nm}
```

where

oplb is the operational label of the medium from which the binary module is to be loaded. The default case for an empty field is GO.

{ident} {nm} ident is an EBCDIC representation of the IDNT of the program to be loaded. It is used for checking purposes only. If nm is specified, it indicates the number of modules to be loaded from oplb; no check of any ident is made. If this parameter is an ident, one module is loaded. If empty, loading proceeds until an EOD is encountered.

DT specifies that calls to M:PUSH, M:PUSHK, M:PUSHC, M:PSHC, and M:RES are modified to dynamic-temp storage (in the calling sequence, "ADRL temp" is changed to "DATA 0", and trailing reserve is stripped). This is done only for those ROMs (including library modules) loaded by this command.

LB The !$LB command controls the search of libraries (for this segment only) to satisfy external references encountered during the loading of modules forming the segment. If the !$LB control command is omitted, the Overlay Loader will first attempt to satisfy all references by definitions in other segments of that path or from the root, and then will search the libraries specified by !$LIB or by the default case. Individual !$LB cards supersede !$LIB or default for that segment only. Libraries are searched only on occurrence of a !$SEG or !EOD control command. !$LIB and !$LB cards only set the mode and sequence of search. Only libraries on the RAD or disk pack may be loaded selectively using the !$LB command. To

input "library" programs from other media, the user must use standard !$LD commands. The format of the command is

```
!$LB  m,x[,y]
```

where

m specifies the search mode and is one of the following EBCDIC codes:

Code	Search Mode
B	Basic (and Main)
E	Extended (and Main)
M	Main only

x[,y] specify the order of library search and are either of the following EBCDIC codes:

Code	Library
S	System
U	User

If y is not specified, only x will be searched.

There are no default values for m, x, or y.

INCLUDE The !$INCLUDE control command specifies external definitions in those library modules that are to be loaded with this segment, even though they are not referenced in the segment. Their definitions will be included in the Symbol Table for use by higher-level segments. More than one !$INCLUDE command may be used. Libraries are searched according to a preceding !$LB or !$LIB card or the initial default case. The format of the command is

```
!$INCLUDE def1[,def2,...,defn]
```

where def_i is an external definition of a library program to be included in the segment.

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 format of the command is

```
!$EXCLUDE  def1,def2,...,defn
```

where def_i is the external definition for a library routine that is not defined along the current path. If def_i is one of several definitions associated with a specific library program, then excluding the one def is sufficient to forestall loading of its associated library module.

MD The !$MD (modify) control command is used to change core locations at load time before the absolute overlays are written out onto the OV file. !$MD commands must be inserted within a SEG sequence and apply only to the segment being loaded. A check is made that the effective address of the !$MD command lies in the segment and that any labels used are defined for the path the segment lies in. The Overlay Loader aborts if the modification location lies outside the limits of the segment. Inserted values are not tested for range. External symbols (definitions) used in loc or value must have been previously defined. The format of the command is

```
!$MD  loc,value[,value1,value2,...,valuen]
```

where

loc specifies the execution location of the first modification, relative to the FWA of the current segment.

$value_i$ is the hexadecimal quantity to be inserted at loc + i (for example, value is inserted at loc, $value_1$ at loc + 1, etc.).

Both the loc and the $value_i$ parameters are subject to the restrictions set forth in "Control Command Format," i.e., hexadecimal notation must be indicated by a leading + or −. Note that it is not possible to modify a library module by use of an !$MD control command.

RES The !RES control command allows the user to reserve an area at the end of the segment (root) program for run-time patching. The format of the command is

```
!$RES def,size[,def,size],...[,def,size]
```

where

def is the name of the area to be reserved.

size is a decimal value specifying the number of words in the reserve area.

LCOM The !$LCOM control command allows the user to allocate labeled COMMON blocks within a segment (root) program. The format of the command is

```
!$LCOM block,size[,block,size]...[,block,size]
```

where

block is the one-to-eight character EBCDIC name of the labeled COMMON block.

size is a decimal value specifying the words to be allocated for the block.

SEG The !$SEG control command defines the modules that will form a segment. Numbers used to define a segment must be unique. Segment identifier numbers need not be consecutive. A segment, including its library, is restricted to a maximum of 65,534 bytes provided enough background is available.

Each !$SEG or !$ROOT control command may be followed by !$LD, !$MD, !$INCLUDE, !$LIB, and !$LB commands to load the modules to form that segment. The loading for a segment terminates on a new !$SEG control command. The control command stack is terminated by an !EOD. The user may defer the loading of a specific library routine through the application of the !$EXCLUDE command. The Loader will attempt to satisfy all references present at a level from the libraries specified on !$LB, !$LIB, and !$INCLUDE commands or from the default library case. A given library is searched only once per segment. The format of the command is

```
!$SEG si,sn[,oplb,n][,DT]
```

where

si is a number less than or equal to X'FF' used to identify the segment being loaded. It will be used to call the segment at run time.

sn is the number of the segment to which this segment is attached.

oplb,n specifies that n modules are to be loaded contiguously from the operational label oplb.

DT specifies that calls to M:PUSH, M:PUSHK, M:PUSHC, M:PSHC, and M:RES are modified to dynamic-temp storage (in the calling sequence, "ADRL temp" is changed to "DATA 0", and trailing reserve is stripped). This is done only for those ROMs (including library modules) loaded by this command.

The following rules should be observed in defining segments for the overlay cluster:

1. In OLOAD, the longest segment must fit into core with the Loader and its tables. If a segment is too long, it may be reassembled as two modules and loaded as two segments.

   In BLOAD, segments (and the root) are loaded one granule at a time, so that all background (less the space required by BLOAD and its tables) is available for symbol tables.

2. The Loader will first attempt to satisfy library references using the Public Library and then will search the appropriate libraries on the RAD or disk pack. Using the !$INCLUDE command, other often-used library routines can be loaded with the root where they will be accessible to all segments. However, library routines loaded in any segment will be accessible only to segments in the same path.

3. Where segment content (not the root) is preceded by reserve area, such area does not consume space during the loading process. However, if a Labeled COMMON block is initially defined by the first module of a segment, it is considered a data area that will precede all reserve areas which will consequently consume space during Loader processing.

PUBLIB (OLOAD only) The !$PUBLIB control command indicates that the Overlay Loader is to create a Public Library using modules that follow and/or modules from selected libraries. The Public Library is biased at the location specified in K:PLFWA of the RBM zero table. Each symbol is flagged as Extended, Basic, or Main according to control information on the !$PUBLIB card. However, a library may contain routines of more than one mode. Such identical definitions of different modes are differentiated in the Symbol Table (LIBSYM) and are not considered duplicate.

When library routines are part of the Public Library, they must be reentrant and therefore must use the dynamic temporary stack (specified as the temp field on the !$ROOT command) for their temporary storage space. The loader will change the calling sequences of any calls to M:RES, M:PUSH, M:PUSHK, M:PUSHC, or M:PSHC to indicate dynamic temporary stack; and will delete trailing reserve from ROMs containing these calls.

A severity level of 1 is set if unsatisfied references or double definitions are encountered during the loading of a Public Library, and the library will not be written onto the PUBLIB file. When a Public Library is being created, the Overlay Loader creates a new Public Library on the RAD or disk pack. The Public Library just loaded is written onto the PUBLIB file in the User Processor area. The total length of the Public Library must not exceed 9191 words. The Monitor Services Transfer Vector (TVECT) file is read from System Processor area, and the Public Library section is updated and written onto TVECT. A new Public Library Symbol Table is written to LIBSYM file in the System Data area. The new LIBSYM is incompatible with the Public Library currently in core. All files are closed and normal termination through M:TERM takes place. The new Public Library is then loaded into core by rebooting the RBM. The format of the command is

```
!$PUBLIB library-mode[,oplb,n]
```

where

mode must be one of the following EBCDIC codes:

Code	Mode
B	Basic
E	Extended
M	Main

A new !$PUBLIB control command must be provided each time mode is to be changed.

oplb,n specifies that n modules are to be loaded contiguously from the operational label oplb.

!$LD, !$LB, !$INCLUDE, and !$MD commands are honored when using !$PUBLIB in the same manner as for the !$SEG command. !$ROOT, !$TCB, and !$SEG commands may not be used in conjunction with the !$PUBLIB command.

END The !$END command is treated exactly like an !EOD command. It should be used in place of !EOD whenever multistep job stacks are to be prestored on a RAD file. The Utility COPY routine will not interpret this command as end-of-file (EOF). The format of the command is

```
!$END
```

LOADER ERROR MESSAGES

The Overlay Loader program outputs messages on both OC and DO concurrently with the load operation. If OC and DO are assigned to the same device, duplication of messages on DO is suppressed. The Overlay Loader error messages are given in Appendix D.

The format of the 'encoded' error messages is

```
$$ ERR xx
```

where xx is a two-letter mnemonic that identifies the error (as described in Appendix D).

The types of Overlay Loader messages are as follows:

1. Warning messages (W), after which loading continues.
2. Response messages (R), requiring an S or X key-in from the operator, in which case the message

```
!!BEGIN WAIT
```

is written on OC. The operator activates the console interrupt and keys in either of the following codes.

Code	Meaning
S	Continue.
X	Abort Overlay Loader with code 'OP' and return control to JCP.

3. Abort messages (A), upon which the Overlay Loader exits via the RBM routine M:ABORT (see also Appendix D for abort codes, abort messages, and their meanings).

The Overlay Loader 'plain text' error messages are largely self-explanatory but are also further described in Appendix D.

8. RAD EDITOR

The RAD Editor controls RAD and disk pack allocation by maintaining file directories for all resident standard areas. A resident standard area is one that has its area mnemonic in the RBM Master Directory (either as a permanent area defined at SYSGEN or a temporary area defined by the Mount key-in) and is not checkpoint (CP), background temporary (BT) area, or of any area whose mnemonic begins with the character X. (X identifies a nonstandard area.) Through its control commands the RAD Editor can

- Add entries to or delete entries from file directories
- Copy data from one random file to another
- Maintain libraries in the system library (SL) and user library (UL) areas for use by the Overlay Loaders
- Copy an object module contained in a library
- Map file and library module allocations
- Dump contents of RAD files, RAD areas, or RAD-type devices in hexadecimal format.
- Save the contents of areas or files in a format restorable by the RAD Editor, or save the contents of areas in a rebootable format on magnetic tape (which may also be restored by the RAD Editor)
- Clear an area or file
- Truncate a file or all files within an area
- Output messages to the operator
- Initialize file directories for new disk packs
- Flaw bad disk pack tracks and allocate alternates.

The RAD Editor generates and maintains directories for the following permanent areas:

- System Processor area (SP)
- System Library area (SL)
- System Data area (SD)
- User Processor area (UP)
- User Library area (UL)
- User Data area (UD and αα)

Size and location of each permanent area are contained in the RBM Master Directory. The RAD Editor allows mapping of all areas, including Checkpoint and Background Temp areas, and the dumping of all random-access files.

STANDARD RAD/DISK PACK AREA ORGANIZATION

Every area contains its own file directory. Each file is identified by a file directory entry that indicates the name, format, and location of the file. The areas and their file directories are software write-protected (at SYSGEN) and may have any of the following four write-protect codes:

Code	Meaning
NO	only files with a write-protect code of NO may be added to the area.
BG	only files with write-protect codes of NO or BG may be added to the area. Background programs may write on any file in the area, but foreground programs may only write on files with NO write-protect codes.
FG	only files with write-protect codes of NO or FG may be added to the area. Foreground programs may write on any file in the area, but background programs may only write on files with NO write-protect codes.
SY	files with any write-protect codes may be added to the area.

For areas with BG or NO write-protect codes, any RAD Editor control command may be used without the need for an SY key-in. However, for areas with FG or SY write-protect codes, the following RAD Edit control commands require that an SY key-in be in effect at the time the control command is executed:

!#ADD

!#DELETE

!#TRUNCATE

!#SQUEEZE

!#RESTORE

!#CLEAR

Space within an area is allocated sequentially; the first file in the area begins in the first sector following the first file directory. The second file in the area begins in the next available sector following the first file. Normally, as each file is added to the area, the next available sector is used as the start of the new file; however, the control command used to allocate space for the file may specify that the file begin on the next available track (or cylinder) boundary. In this event, any space bypassed will remain unused and the RAD Editor will not attempt to fit a new file into the unused space. New files are always added at the end of the currently allocated space within an area.

When a directory entry (and, effectively, its corresponding file) is deleted, the area formerly occupied by the file is left unused. In normal operation, the RAD Editor makes no attempt to recover these unused areas. Therefore, the addition of a file may cause overflow of the permanent area although ample space may be available. However, RAD squeezing can be requested via an Editor !#SQUEEZE command to overcome this problem. Squeezing recovers the unused storage within a permanent area by regenerating the directory and moving files.

Before any permanent file can be written (using the Monitor routine M:WRITE), space must be allocated for the file. This is accomplished by requesting the RAD Editor to add a new entry ot the designated directory. Control commands allow directory entries to be added or deleted.

Warning: While processing the commands ADD, DELETE, TRUNCATE, SQUEEZE and CLEAR, foreground files that may become active are the user's responsibility.

DATA FILES

Ordinarily, data is not written in permanent files by the RAD Editor. Data files are normally written by user programs. However, a RAD Editor control command can be used to copy data from one random-access file to another. Copied files may be temporary or permanent files.

LIBRARY FILES

System and User Library files, which are searched by the Overlay Loaders for external references, are generated and maintained by the RAD Editor (the only processor that writes in these files).

A library area (either the System Library area or the User Library area) contains six files:

1. Module Directory File (directory of library modules).
2. EBCDIC File (list of all library definitions/references).
3. Extended DEF/REF File (index to extended precision definitions/references in EBCDIC file).
4. Basic DEF/REF File (index to standard precision definitions/references in EBCDIC file).
5. Main DEF/REF File (index to main definitions/references in EBCDIC file).
6. Module File (library object modules).

The extended and basic DEF/REF files (items 3 and 4) are optional.

These files are generated and maintained from information in control commands and object modules placed in the library by the RAD Editor. Special commands are supplied to allow the addition and deletion of object modules; these control commands will cause the six files in the RAD Library area to be updated. A control command allows an object module contained in a library to be copied onto BO.

Any random-access or sequential-access file (either temporary or permanent) can be dumped on LO.

The RAD Editor can save the contents of a permanent area and the RBM bootstrap in a self-reloadable form. The saved image contains a bootstrap loader, the execution of which restores the RBM bootstrap and the permanent area on the RAD or disk pack.

Updating or squeezing of permanent areas and library files that contain information for real-time programs must not occur while the foreground is using these permanent areas or files. The user must ensure that the RAD Editor is not modifying a permanent area while a foreground program is using it.

The names for the library files must be one of the following:

Code	File
MODIR	Module Directory
EBCDIC	EBCDIC
EDFRF	Extended DEF/REF (optional)
BDFRF	Basic DEF/REF (optional)
MDFRF	Main DEF/REF
MODULE	Module

The DEF/REF file needs to be added only as required. The System Library (SL) requires only the MDFRF file.

ALGORITHMS FOR COMPUTING LIBRARY FILE SIZES

The following algorithms may be used to determine the lengths of the six files in a library area:

The number of granules in the MODIR file is

$$MODIR_n = \frac{6\,(1 + i)}{g}$$

where

- i is the number of modules to be placed in the library (including main, extended-precision, and single-precision routines). i must be equal-to or less-than 1023.
- g is the granule size in words.

The number of granules in the EBCDIC file is

$$EBCDIC_n = \frac{4\,(1 + d)}{g}$$

where

d is the number of unique DEFs and REFs in the library (including main, extended-precision, and single-precision routines). d must be equal-to or less-than 8191.

g is the granule size in words.

The number of granules in the EDFRF file is

$$EDFRF_n = \frac{2 + \sum_{\ell=1}^{n}(2 + r_\ell + d_\ell)}{g}$$

where

n is the number of routines in the extended-precision library.

r_ℓ is the number of REFs in the extended-precision library.

d_ℓ is the number of DEFs in the extended-precision library.

g is the granule size in words.

The number of granules in the BDFRF file is

$$BDFRF_n = \frac{2 + \sum_{k=1}^{n}(2 + r_k + d_k)}{g}$$

where

n is the number of routines in the single-precision library.

r_k is the number of REFs in the kth library routine in the single-precision library.

d_k is the number of DEFs in the kth library routine of the single-precision library.

g is the granule size in words.

The number of granules in the MDFRF file is

$$MDRFR_n = \frac{2 + \sum_{j=1}^{n}(2 + r_j + d_j)}{g}$$

where

n is the number of routines in the main library.

r_j is the number of REFs in the jth library routine in the main library.

d_j is the number of DEFs in the jth library routine in the main library.

g is the granule size in words.

The number of records in the MODULE file is

$$MODULE_n = \sum_{i=1}^{n}(c_i)$$

where

n is the number of modules in the library (including main, extended-precision, and single-precision routines), and n must be equal to or less than 1023.

c_i is the number of record images in the ith library routine.

RAD EDITOR OPERATIONAL LABELS

The RAD Editor uses the temporary background operational labels X0 through X6. These labels must not be assigned at the time the !RADEDIT control command is executed, nor may they be used on !#DUMP or !#FCOPY commands.

The following labels must be assigned before requesting the RAD Editor:

Label	Explanation
BI	Object module input (and Restore) to System and User Library.
BO	Object module output (and Save) from the System and User Libraries.
CC	Control command input. If KP is in effect, control command input is read from oplabel 'OC'.

Label	Explanation
DO	Log of error messages and operator key-ins.
LL	Log of control commands.
LO	Maps of directories and dumps of files.
OC	Messages to the operator and key-ins from the operator. Control commands are read from OC if a "KP" key-in is in effect.

CALLING RAD EDITOR

The RAD Editor is requested with a !RADEDIT control command. The !RADEDIT control command is read from CC and causes the root segment of the RAD Editor program to be loaded into core memory from the RAD. It has the format

```
!RADEDIT
```

Reading an !EOD from CC causes the RAD Editor program to return control to the Monitor. If CC is assigned to magnetic tape or a RAD file, an EOF condition encountered while reading control commands from CC will cause the RAD Editor to return control to the Monitor. The form of the command is

```
!EOD
```

CONTROL COMMAND FORMAT

All RAD Editor control commands are input from CC (or from OC if a "KP" key-in is in effect) and listed on LL. The general format is

```
!#menmonic specification
```

where

! identifies the record as a control command.

{#, =} indicates that the control command is unique to the RAD Editor.

mnemonic is the code name of a RAD Editor command immediately following the !# characters.

specification is a series of required or optional parameters unique to the specific command. The conventions used in specifying parameters are (1) a string of up to five decimal digits, having a value less than 65,535, denotes a decimal integer; (2) a string of the form +xxxx is treated as hexadecimal; (3) all other strings are assumed to be nonnumeric.

One or more blanks must separate the mnemonic and specification fields, but no blanks may be embeddded within a field. An empty parameter in the specification field is denoted by a comma. However, commas may be omitted for empty trailing parameters. A control command is terminated by the first blank after the specification field. If the specification field is absent and a comment follows the mnemonic field, the command is terminated by a period.

The first two characters of the mnemonic portion of the command are sufficient to define the command; the remaining characters may be omitted since they are ignored if they are present.

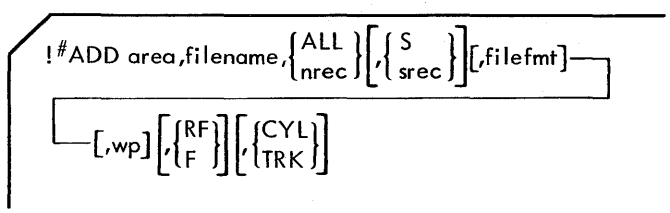
In the descriptions of the following individual commands, certain terms are used that have specific meanings for the RAD Editor. The terms are:

Term	Meaning
area	The two-character alphanumeric mnemonic for a resident standard area. The area mnemonic must be currently present in the RBM Master Directory and, generally, may not be BT, CP, or Xn. For the commands !#LADD, !#LREPLACE, !#LDELETE, !#LCOPY, !#LMAP, and !#LSQUEEZE, area must be either SL or UL. If neither is specified, SL is assumed by default.
filename	Three to eight alphanumeric characters denoting a file contained within (or to be added to) an area file directory. At least one character must be alphabetic.
identification	The library routine name denoted by the Extended Symbol IDNT directive, which is located in the start module load item of an object module.
library	An object module library (within the System or User Library) denoted by one of the codes Code — Library M — Main E — Extended B — Basic For the commands !#LADD, !#LREPLACE, and !#LDELETE the default library is M (main).

CONTROL COMMAND REPERTOIRE

ADD The !#ADD command adds a new entry to the specified permanent file directory. It defines the name,

size, record length, format, and write protection for the new file. It may also declare that the file will contain a resident foreground program, and will be maintained starting at a cylinder or track boundary. Space is allocated for the new file and the first sector of the file is set to zero if it has random format. The form of the command is



where

ALL indicates the file will be allocated to extend to the end of the area. After an EOF has been written on the file, it may be truncated to recover the unused space.

nrec is the number of records in the file and may not exceed 65,535.

S indicates that the record size is equivalent to sector size and that nrec is to be used to determine the number of sectors to reserve.

srec is the maximum number of bytes per record which must be even and may not exceed 65,534. If filefmt is R, srec is used as the granule size. The following default values are provided, depending on the file format.

Default Record Size

- 120 for file format, B and P
- Sector size, in bytes, of the device containing the area for file format R or U
- 80 for file format C. Since compressed files may contain records of variable length, this value is used for allocation purposes only. The S character may be used to force the allocation of a specific number of sectors for a compressed file. In this case, nrec indicates the number of sectors to reserve for the compressed file.

filefmt is the structure of the file, as denoted by one of the following codes:

Code	Format
B	blocked sequential-access file with a fixed record size
C	blocked (and compressed) sequential access file with a variable record size
P	blocked (packed) random access file, fixed record size
R	unblocked random access file
U	unblocked sequential access file.

If the format parameter is omitted, the default format is determined by the area mnemonic as follows:

Default	Area Mnemonic
R	SP, SL, UP, UL, FP, BP
B	any other

wp specifies the write-protection level for the file, as denoted by one of the following codes:

Codes	Write-Protection Level
NO (or N)	No write-protection; background or foreground programs may write on the file.
BG (or B)	Write permitted by background programs only.
FG (or F)	Write permitted by foreground programs only. Background programs may write on the file if an SY key-in is in effect.
SY (or R)	Write permitted by RBM only. Foreground or background programs may write on the file if an SY key-in is in effect.

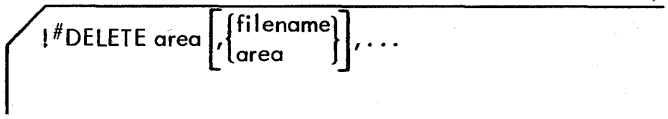
If the wp parameter is omitted, the default write-protection level is NO.

RF or F specifies that the file will contain a resident foreground program that is to be automatically loaded at boot time, and therefore the area mnemonic must be SP, UP, or FP (the order of search).

CYL specifies that the BOT of the file is to be allocated and maintained on a cylinder boundary if the area is on a disk pack.

TRK specifies that the BOT of the file is to be allocated and maintained on a track boundary.

DELETE The !#DELETE command deletes on entry from the specified permanent file directory. The space formerly allocated to the file becomes unused. The space is recovered if the file being deleted is the last file in the area. The Form of the command is



If no filename is specified, all files in the area are deleted. If there are active files in the area, the operation is not performed. Under no condition can the SP area be deleted. Instead, the following messages are output

```
##OPEN FILES, NO CHANGE: area, filename
```

```
## NO CHANGE: area
```

If the write-protect code for the area is SY or FG, the SY key-in must be in effect at the time the control command is executed.

FCOPY The !#FCOPY (File Copy) command copies data from one random-access file to another. The file copy process terminates when EOF is encountered on an input file or when an end-of-tape is encountered on either the input or the output file. The form of the command is

```
!#FCOPY oplb1,oplb2
```

where

$oplb_i$ is the operational label or FORTRAN device unit number (e.g., F:109) of a temporary or permanent random-access file. The Utility COPY Routine (see Chapter 9) must be used to copy sequential-access files.

$oplb_1$ is the input file.

$oplb_2$ is the output file.

DPCOPY The !#DPCOPY (Disk Pack Copy) control command copies data from one disk park or cartridge disk to another. The entire contents of the pack or cartridge is copied and a checkwrite is performed on the copied data. The form of the command is

```
!#DPCOPY +device1,+device2
```

where

$device_1$ is the hexadecimal device number of the disk pack

$device_2$ is the hexadecimal device number of the disk pack to copy to, which may not contain any currently "mounted" areas.

Note: The bootstrap record is not copied to sector zero. The INITIALIZE command always writes the bootstrap record to sector zero.

LADD The !#LADD (Library Add) command adds an object module to the designated library. The object module is read from BI, checked for sequence and checksum errors, and stored in the Module File within the library. From the data in the object module and on the control command, the information about the module is extracted and placed in the Module Directory File (MODIR), the EBCDIC File, and one of the three DEF/REF Files (either MDFRF, BDFRF, or EDFRF File) as indicated in the library parameter. BI may be assigned to any device; if BI is assigned to the RAD, it must be sequential file. The object module on BI must be in standard object language. Any blank card or binary card on BI that contains only zeros is ignored. The form of the command is

```
!#LADD [area,][ident][,library]
```

where ident is the program name located in the start module item of the object module on BI.

The library routine may be selectively added to the SL or UL area from a file of library routines. If the identification parameter is omitted, all object modules on BI will be added to the library up to, but not including, the file mark or EOD on BI.

Within a permanent area (SL or UL), each object module ident must be unique except as follows: an object module ident in the Main library cannot exist in either the Basic or Extended library. An object module with the same ident can exist in both the Basic and Extended libraries.

If identification is present, the start module load item of the first program read from BI must be the same as shown by the identification parameter.

LREPLACE The !#LREPLACE (Library Replace) command replaces an object module of the same identification in the designated library. The object module is read from BI and checked for sequence checksum errors. The object module on BI must be in standard object language. Any blank card or binary card (on BI) that contains only zeros is ignored. The form of the command is

```
!#LREPLACE [area,]ident[,library]
```

where ident is the program name located in the start module item of the object module on BI. The object module on BI replaces the module in the library having the same identification.

LDELETE The !#LDELETE (Library Delete) command deletes an object module from the designated library. The form of the command is

```
!#LDELETE[area,]ident[,library]
```

where ident is the program name of the object module to be deleted.

LCOPY The !#LCOPY (Library Copy) command copies an object module from the designated library onto the BO device. The form of the command is

```
!#LCOPY [area,]ident
```

where ident is the program name (located in the start module item) of the object module to be copied onto the BO device.

LSQUEEZE The !#LSQUEEZE (Library Squeeze) command will squeeze designated library areas. Unused space is recovered by regenerating the directory files and

squeezing (compacting) the module file. The form of the command is

```
!#LSQUEEZE [area]
```

MAP The !#MAP command causes the specified directories to be mapped on LO. For each permanent RAD area, the beginning and ending RAD addresses for the area are mapped. For each file, the contents of the directory entry describing the file are printed. This information includes name, format, write-protection, foreground task indicator, beginning address, EOF address, and EOT address for each file. For files on disk packs, the map also includes the cylinder/track/sector values for BOT. For files on RADs, the map also includes the track/sector values for BOT. The form of the command is

```
!#MAP [area1] [,area2]... [,arean]
```

where area must be a currently defined area. If no area parameter is included, all currently defined areas are mapped.

LMAP The !#LMAP command causes the library files of the specified areas to be mapped on LO. For each area, the beginning and ending addresses for the area are mapped, followed by a map of the library files in the area. The library map includes the following information for each routine:

- Library B (basic), E (extended) or M (main)
- Identification of routine
- Length of routine in words
- Sector within the MODULE file that contains the routine
- DEFs in the routine
- REFs in the routine.

The form of the command is

```
!#LMAP [area [,area]]
```

DUMP The !#DUMP command dumps a RAD file, a RAD area, or a RAD-type device in hexadecimal format on the LO device. Records are dumped beginning with the first record of the file (record 0) unless an optional starting record number is given. The dump is terminated by an EOF, EOT, or by having dumped the requested number of records – whichever occurs first. The form of the command is

```
!#DUMP {oplbl                          } [, start [, number]]
       {area [, filename]              }
       {+device-number                 }
```

where

oplbl is the operational label or FORTRAN device unit number (e.g., F:109) assigned to a RAD file. Operational labels X0 through X6 may not be used because they are reserved for use by RADEDIT.

device-number is the hardware device number of the RAD or disk device. This number must be preceded by a + (plus character) and must match a RAD or disk device number input at SYSGEN.

start is the relative record (or sector) number at which the dump is to begin. For RAD files, this number represents the record relative to the first record of the file. For RAD areas, it represents the sector relative to the first sector of the area. For device-number, it represents the sector relative to the first sector of the device. If start is omitted, the dump begins at the first record relative to the BOT of the file, area, or device.

number is the number of records (or sectors) to be dumped. If number is omitted, the file, area, or device is dumped until an EOF or EOT is encountered. If the file format is random (R) or packed (P), the EOF is ignored.

SAVE The !#SAVE command saves the contents of areas of specific files. Each file is written on the BO device, along with all pertinent information about the file. The BO media may be magnetic tape, unblocked file, paper tape, or cards. If the media is magnetic tape and an end-of-reel condition occurs, the operator is expected to mount the next reel to be used for output. If the media is paper tape and an !ATTEND command has been input for the current job, the message

```
## nnnn FT. OK?
```

will be output on the OC device. If there is more than nnnn feet of paper tape available, the operator is expected to type in Y. This process will continue until all files specified by the !#SAVE command have been output, or until the operator determines that the required amount of tape is not available. Any input other than Y causes the program to output an end-of-reel record followed by blank trailer. The program then outputs the message

```
!!BEGIN WAIT
```

on the OC device. The operator must then mount a new reel of tape, interrupt, and key-in S. The program then outputs blank leader, a save-continuation record, and proceeds as described above.

The BO output can be restored via the !# RESTORE command. The form of the command is

```
!#SAVE [FILE,][area][,{area|filename},...],...
```

where FILE indicates that the output format contains all necessary information for the restoration of specific files. Each file saved is followed by an !EOD (or file mark in the case of magnetic tape). If another !#SAVE FILE control command follows immediately, the additional files are appended to the previous output.

Note: !#SAVE FILE command does not save the following files from the SP and SD area:

SP, BOOT
SP, RBM
SP, TVECT
SP, RADEDIT
SP, OLOAD
SD, RBMGO
SD, RBMOV
SD, RBMAL
SD, ERRFILE
SD, RBMID
SD, RBMSYM
SD, RBMPMD

When a control command is encountered that is not a !#SAVE FILE command, an additional EOF is written and the BO device is rewound before the next command is executed.

The FILE keyword may be omitted only if the BO operational label is assigned to magnetic tape, causing a bootstrap program to be output on BO followed by the contents of the specified areas. No filenames may be specified in this case, since the allocated portion of an area is saved as if it were a single file. The specified areas are followed by two file marks and the tape is rewound.

When the output magnetic tape is booted, the bootstrap program will restore the saved areas and then initiate an RBM boot process.

The user must not mix the output of the !#SAVE command with the !#SAVE FILE command on the same magnetic tape.

VERIFY The !#VERIFY command checks the output of the !#SAVE command to ensure that it can be correctly processed by !#RESTORE at a later time. The form of the command is

```
!#VERIFY [oplbl]
```

where oplbl is the operational label of the medium from which the !#SAVE output is to be read. If no oplbl is present, the oplbl BO is assumed by default.

RESTORE The !#RESTORE command restore the contents of areas or specific files that have been saved via the !#SAVE command. The files are selectively restored from the BI device. The form of the command is

```
!#RESTORE [area[,{area|filename},...],...]
```

If the file being restored does not have a corresponding entry in the area file directory, a new entry is made and the file is copied into its allocated region. If the file being restored already has an entry in the area file directory the file will be copied into the currently allocated region unless

- There is a format conflict
- The allocated region is too small
- The proper level of write authorization is not in effect, that is, SY key-in not performed and file is write-protected.

If an end-of-reel condition is encountered while reading from BI the operator will be requested to mount the next reel in sequence, as created by the !#SAVE command.

If the BI input is a rebootable save tape, no filenames may be specified – each area is restored in its entirety.

SQUEEZE The !#SQUEEZE command compacts the designated file areas. Unused space is regained by regenerating the directories and moving files. The form of the command is

```
!#SQUEEZE area[,area]...[,area]
```

The areas BT, CP, and any area beginning with the letter X are never squeezed. An explicit request to squeeze any of these is ignored. If the area being squeezed contains a file that is assigned to an operational label and the file can be moved, the following message will be output.

```
## OPEN FILE, NO CHANGE: area, filename
```

File is not moved and squeezing continues.

File directory information may be destroyed if an area being squeezed contains a file that is assigned to an operational label being used by an active foreground program.

Care should be exercised when SQUEEZEing areas that may be currently in use by foreground programs in order to avoid file conflicts.

CLEAR The !#CLEAR command zeros out the specified RAD area or file. The form of the command is

```
!#CLEAR area[,{area|filename}],...
```

If no filename is specified, all files in the area are cleared, including file directories. If there are open files in the area, the operation will not be performed. Instead, the following message will be output

```
## OPEN FILE, NO CHANGE: area, filename
```

If the write-protect code for the area is SY or FG, the SY key-in must be in effect at the time the control command is executed.

BDTRACK The !#BDTRACK command specifies the disk pack and the track numbers for which alternates are to be provided.

Two methods of selecting alternate tracks are used: the flawed headers and the bad track list.

The disk packs, Models 7242/46, use flawed headers. The original track will have its headers rewritten with a flaw mark and a reference to the alternate track. The headers of the alternate track will be rewritten to refer to the original track. The cartridge disks, Models 3231/32/33 and 7251/52, utilize a bad track list which is written on cylinder 0, track 0, sector 2. The bad track numbers are written into the list and the corresponding alternate tracks are selected during the read-write process. The form of the command is:

```
!#BDTRACK +dn, {ALL
               +number[, +number]...[, +number]
               decimal[, decimal]...[, decimal]}
```

where

dn is the device number of the disk pack.

number is the hexadecimal track number on the device starting with 0.

decimal is the decimal track number on the device starting with 0.

ALL indicates that a bad track list is to be constructed from the flawed headers previously written on the 323x device. Once this is done, !#BDTRACK commands can be used to enter track numbers into the bad track list on the device.

Note: See the Unsolicited Control section as to how bad track lists are entered (Mount) and removed (Remove) from the system tables.

Example:

```
!#BDTRACK +E5, +325, +297
```

GDTRACK The !#GDTRACK command specifies the disk pack and the track numbers for which alternates are to be eliminated. This may be used if it is suspected that a designated flawed track is good.

On the disk packs, Models 7242/46, for each track specified, its headers will be rewritten to clear the flaw mark and the headers of the assigned alternate track will be rewritten to free the alternate track.

On the cartridge disks, Models 7251/52 and 3231/32/33, which utilize a bad track list, a "blank" bad track list can be written on cylinder 0, track 0, sector 2 of the device using the "ALL" option. If a bad track list exists on the device, bad tracks can be eliminated as described above. The form of the command is:

```
!#GDTRACK +dn {ALL
               +number[, +number]...[, +number]
               decimal[, decimal]...[, decimal]}
```

where

dn is the device number of the disk pack.

number is the hexadecimal track number on the device starting with 0.

decimal is the decimal track number on the device starting with 0.

ALL indicates that a "blank" bad track list is to be written on the device.

INITIALIZE The !#INITIALIZE command provides disk pack serialization (including date) and allocation of data areas. The form of the command is

```
!#INITIALIZE +dn[, serial-number], DO
```

where

dn is the hardware device number of the disk pack to be initialized. The device number must match a disk pack device number input at SYSGEN.

serial number is any combination of eight characters, excluding blanks or commas. If the disk pack is Xerox Model 7242 or 7246, the serial number is written on cylinder 202, track 19, sector 0, together with the current date. For Xerox Models 725x and 323x, the serial number and date are written on cylinder 0, track 0, sector 1.

DO indicates that the file directory on the device is not to be initialized.

The !#INITIALIZE command may be followed by a set of area definition cards that have the format

```
!#area=tracks[,wp]
```

where

area is an area mnemonic from the following list:

SP	BP	SL	UP
SD	CP	BT	UL
FP	SK (skip tracks)	αα	

tracks is the number of tracks to be allocated for the area. A parameter of 'ALL' allocates all the remaining tracks on the device.

wp is the write-protect code to be used for the area. This code is tested whenever any of the following operations are performed:

ADD	DELETE	SQUEEZE
CLEAR	RESTORE	TRUNCATE

See "Standard RAD/Disk Pack Area Organization" for write-protect codes.

MESSAGE The !#MESSAGE control command writes messages to the operator on the OC and LO devices. The form of the command is

```
!#MESSAGE message
```

where message is any EBCDIC character string up to a full card image.

PAUSE The !#PAUSE control command causes a message to be written on the OC and LO devices followed by a wait for the operator's response. The form of the command is

```
!#PAUSE message
```

where message is any EBCDIC character string up to a full card image. The format of the output is:

```
!#PAUSE message
!!BEGIN WAIT
```

It is necessary for the operator to activate the control panel INTERRUPT switch and key-in an S to continue.

TRUNCATE The !#TRUNCATE command eliminates unused space from the end of specific files by setting the EOT pointer equal to the EOF pointer. If an EOF has not been written on the file, the file EOT will not be changed. All compressed files or files containing programs loaded by the Overlay Loaders (or with the Monitor !ABS command) will have an EOF pointer. The space is recovered if the file being truncated is the last file in the area. The form of the command is

```
!#TRUNCATE area [,{area | filename}],...
```

If no filename is specified all files in the area containing an EOF are to be truncated to the EOF. If the area is SD and no filename is specified, the following message will be output on the DO device and the OC device (if KP is in effect) and providing oplabels are not assigned to same device.

```
## NO CHANGE: SD
```

If the write protect code for the area is SY or FG, the SY key-in must be in effect at the time the control command is executed.

END The !#END command is used exactly like the !EOD command; that is, it transfers control from the RAD Editor to the Monitor. The form of the command is

```
!#END
```

This command should be used in place of !EOD whenever multistep job stacks are to be prestored on a file. The Utility COPY routine will not interpret this command as an EOF.

RAD EDITOR MESSAGES

The RAD Editor program issues the error messages listed in Appendix D (Table D-4) on DO. If a KP key-in is in effect, the error message is output on OC and DO unless OC and DO are assigned to the same device. The warning (W) messages in Table D-4 are written on the OC and DO devices to provide a record of operations not performed or of critical operations in process. If an operator response is required, the R-type error message is followed by the RBM message.

```
!!BEGIN WAIT
```

written on OC. The operator activates the PCP interrupt and keys in one of the following:

Key-In	Meaning
SY, S	Suspend disk write-protect and continue.
S	Continue.
X	Abort RAD Editor and return control to RBM.

RAD Editor initiated aborts are identified by the abort code 'RE'. If the abort is operator-initiated, this is indicated by the abort code 'OP'.

9. UTILITY

INTRODUCTION

The Utility program operates in the background under the Real-Time Batch Monitor. It contains routines that:

- Copy variable-length binary or EBCDIC records from one medium to another (Copy).
- Dump records onto an output device in either hexadecimal or EBCDIC format (Dump).
- Generate or update files that contain Xerox Standard Object Language modules (Object Module Editor).
- Generate or update symbolic files (paper or magnetic) that contain source data (Record Editor).
- Edit card images by sequence number (Sequence Editor).

Routines in the Utility program are device-independent. Utility handles any blocked or unblocked, sequential-access RAD file. Use of a sequential-access RAD file is similar to that of a magnetic tape, as it has a beginning-of-tape, an end-of-file (if one has been written), and an end-of-tape. Note, however, that a sequential-access RAD file cannot be forward-spaced or backspaced over more than one file mark. A rewound sequential-access RAD file is positioned at beginning-of-tape. For both blocked and unblocked files, a record skip is a logical record skip.

UTILITY PROGRAM ORGANIZATION

The Utility program consists of two major sections: the Utility Program Control routine (always resident when the Utility program is operating), and the currently operating Utility subroutine. The Utility Program Control routine contains four interdependent elements:

1. The Program Executive, which initializes the program (upon entry from RBM), interprets the !UTILITY control command (explained in "Calling Utility"), exercises control over the flow of control commands, handles normal and abort exits to the Monitor, and performs all I/O checking for the Utility program.
2. The Source Input Interpreter, which reads and scans Utility control commands for the Control Function Processor and the current Utility subroutine.
3. The Control Function Processor, which executes control function commands common to all Utility subroutines.
4. The Operator Communication routine, which outputs messages to OC and DO and receives key-in responses.

UTILITY PROGRAM EXECUTIVE

When RBM reads a !UTILITY control command control is transferred to the Program Executive routine. The !UTILITY control command is then scanned for parameters. If the name parameter is omitted (see "Calling Utility" below), it is assumed that only the Control Function Processor will be used. Utility control commands are read from the source input (SI) device unless a KP key-in is in effect, in which case commands are read from OC.

If a specific Utility subroutine is requested, the Program Executive verifies that the subroutine is in storage; if not, an error message is written and an exit to RBM is taken, terminating the background operation. If the subroutine is present, initialization of tables and flags occurs.

The Program Executive then transfers control to the requested Utility subroutine. The Utility subroutine uses the Source Input Interpreter to read all commands, and uses the Control Function Processor to execute control functions. All other control commands are interpreted and executed by the Utility subroutine itself.

SOURCE INPUT INTERPRETER

The Source Input Interpreter, which is called by the Program Executive routine, processes all control commands that are read by the Utility program. Utility control commands are input from the SI device (unless a KP key-in is in effect) and listed on the LL device as they are interpreted.

Upon reading a command, the Source Input Interpreter determines whether the command is valid. If the syntax for a command is invalid, the following message is written on OC and DO:

```
** INV CTL
```

The Utility program then reads the next command or enters the wait state if attend mode is in effect.

If the command is valid, it may be interpreted and executed either by the Utility subroutine or by the Control Function Processor.

CONTROL FUNCTION PROCESSOR

The Control Function Processor interprets and executes commands that are common to all Utility subroutines. If any of

the control commands interpreted and executed by the Control Function Processor contains an invalid operational label, the following message is output:

```
** INV OPLB
```

The Utility program then reads the next command or enters the wait state if attend mode is in effect.

CONTROL ROUTINE OPERATIONAL LABELS

Four operational labels are reserved for the Program Control routine. Their use is restricted to the functions below; they may not be used in place of the labels required by the various Utility subroutines explained later.

Label	Explanation
SI	Device for Utility control command input, and various modification source inputs. (If a KP key-in is in effect, control commands are read from OC.)
DO	Device for listing of messages, error conditions, operator responses, etc. If OC and DO are assigned to the same device, duplication of messages is suppressed.
LL	Log of control commands.
OC	Device for messages to the operator, key-in responses from the operator (always via the keyboard/printer), and control-command input if a KP key-in is in effect.
X5	Temporary RAD file used for prestoring commands read from SI.

Utility functions are generally executed dynamically; that is, control commands are interpreted and executed as they are read. However, when several operational labels are assigned to the same device-file as SI, it is impractical to execute dynamically. In this case, commands must be prestored to avoid confusion with data from that device. This decision to prestore is made by the Utility program with one exception: the !*PRESTORE control command allows the user the option of prestoring control command input until an EOD card image is encountered. For RBM Utilities, prestored commands are written on a temporary RAD file (using operational label X5) and read from the RAD for interpretation and execution.

CALLING UTILITY

The Utility program is requested via a !UTILITY control command, which causes the root segment of the Utility program to be loaded into core memory from the RAD. The !UTILITY control command has the format

```
!UTILITY [name][,parameter]
```

where

name is the name of a Utility routine or may be omitted. It may be any of the following:

COPY	(Copy)
DUMP	(Dump)
OMEDIT	(Object Module Editor)
RECEDIT	(Record Editor)
SEQEDIT†	(Sequence Editor)

parameter represents the series of optional parameters that are unique to each Utility routine. Parameters are fully explained in the description of the individual routines.

When RBM reads the !UTILITY command, it loads the Program Control routine (root segment) from the RAD and transfers control to the Program Executive which controls the operation of the Utility program. The Executive first scans the !UTILITY control command parameters. If the name parameter is omitted, the Executive assumes that the control commands that follow use the Control Function Processor only. If a specific Utility routine is referenced with the name parameter, the Program Executive checks the name for validity. If the name is invalid, the message

```
** UT  NT  RES
```

(Utility not resident) is written on OC and DO and the Utility program aborts. If the name is valid, the overlay segment containing the Utility routine is loaded from the RAD, flags are initialized, and control is transferred to the named routine.

When the Program Control routine encounters an !EOD card image from SI, it terminates processing. The form of the !EOD command is

```
!EOD
```

This causes the Utility program to transfer control back to RBM.

---

†The Sequence Editor always reads from SI, whether or not a KP key-in is in effect.

If a Utility routine encounters the control command !UT [name] [,parameter], normal termination occurs and the named routine is loaded and given control without return to RBM.

CONTROL COMMAND FORMAT

All Utility program control commands are input from SI and are listed on the LL device as they are interpreted. The general format is

```
!*mnemonic specification
```

where

- ! identifies the record as a control command.
- * indicates that the control command is unique to the Utility program.
- mnemonic is the code name of a Utility command and begins immediately following the !* characters.
- specification is a series of parameters unique to the specific command. The conventions used in specifying parameters are (1) a string of up to five decimal digits having a value less than 32,768 denotes a decimal integer and (2) a string containing more than five characters is always assumed to be EBCDIC, regardless of content.

One or more blanks separate the mnemonic and specification fields, but no blanks may 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 mnemonic field, the command is terminated by a period. No control command record may contain more than 80 characters. The first two characters of the mnemonic portion of the command are sufficient to define a control command; the remaining characters may be omitted, since they are ignored when present.

CONTROL FUNCTION COMMANDS

The Control Function Processor interprets and executes control commands that are common to all Utility subroutines. These control function commands are given below. Unless otherwise noted, "oplb" is the operational label of the device, "number" is the number of file marks or records to skip (if omitted; the number is assumed to be 1), and "device" is the device type and physical device number.

FBACK The !*FBACK command backspaces a magnetic tape over a specified number of file marks or a sequential-access RAD file to beginning-of-tape (BOT). The form of the command is

```
!*FBACK oplb[,number]
```

The !*FBACK command cannot be used for random files.

FSKIP The !*FSKIP command spaces a magnetic tape forward over a specified number of file marks or a sequential-access RAD file over its end-of-file. The form of the command is

```
!*FSKIP oplb[,number]
```

The !*FSKIP command cannot be used for random files.

MESSAGE The !*MESSAGE command writes messages to the operator on the OC and the DO devices. The form of the command is

```
!*MESSAGE message
```

where message is any EBCDIC character string up to a full card image.

The format of the output is

```
!*MESSAGE message
```

PAUSE The !*PAUSE command causes a message to be written on the OC and DO device followed by a wait for the operator's response. The form of the command is

```
!*PAUSE message
```

where message is any EBCDIC character string up to a full card image.

The format of the output is

```
!*PAUSE message
!!BEGIN WAIT
```

PRESTORE The !*PRESTORE command causes all control commands to be read from the SI device, but not to be interpreted or executed until an !EOD is read. The prestored commands are written on a temporary RAD file (using operational label X5) and are read sequentially from the RAD. (The prestore mode is set automatically when a name parameter appears on the !UTILITY command and one or more operational labels have been assigned to the same device or RAD DFN as SI.) The !*PRESTORE control command must immediately follow the !UTILITY control command and must precede any other control commands for the Utility program. The form of the command is

```
!*PRESTORE
```

REWIND The !*REWIND command causes the specified magnetic tape or sequential-access RAD file to be rewound. The form of the command is

```
!*REWIND oplb
```

RBACK The !*RBACK command backspaces a magnetic tape or sequential-access RAD file over a specified number of records. The form of the command is

```
!*RBACK oplb[,number]
```

If oplb is assigned to a blocked sequential-access RAD file, the number parameter is the number of logical records to be skipped. The !*RBACK command cannot be used for random files or compressed RAD files.

RSKIP The !*RSKIP command spaces forward the indicated magnetic tape or sequential-access RAD file over the specified number of records. The form of the command is

```
!*RSKIP oplb[,number]
```

If oplb is assigned to a blocked sequential-access RAD file, the number parameter is the number of logical records to skip. The !*RSKIP command cannot be used for random files but can be used for compressed RAD files.

UNLOAD The !*UNLOAD command unloads a magnetic tape or closes a sequential-access RAD file. The form of the command is

```
!*UNLOAD oplb
```

END The !*END command is treated exactly like an !EOD; that is, transfers control from Utility to the Monitor. This command should be used in place of !EOD whenever multiactivity job stacks are to be prestored on a RAD file. This command will not be interpreted as an EOF when read from UI. The form of the command is

```
!*END
```

WEOF The !*WEOF command writes a file mark, EOD, or end-of-file pointer if appropriate to the device. The form of the command is

```
!*WEOF oplb
```

ASSIGN The !*ASSIGN command allows a Utility user to assign any operational label to any other background operational label, device-file number, or RAD file. The form of the command is

```
!*ASSIGN oplb{=,} {oplb
                  dfn
                  file,area
                  fdun}
```

where

dfn is a device-file number.

file is a RAD file name.

area is the RAD area within which the RAD file is defined.

fdun is a FORTRAN device unit number.

COPY ROUTINE

COPY provides the ability to copy variable-length binary or EBCDIC records from cards, paper tape, magnetic tape, keyboard/printer, and sequential-access RAD files to cards, paper tape, magnetic tape, line printer, keyboard/printer, and sequential-access RAD files. Using control functions of the Control Function Processor, records and files can be skipped except for random files. The COPY routine also provides for file verification (separate from the copy operation). If the binary mode is requested for either copying or verifying, file marks are recognized for paper tape, magnetic tape or sequential RAD file. An !EOD card is recognized as a file mark. The number of records and files read or verified is listed upon completion of the COPY or VERIFY operation.

Since COPY uses RBM routines M:READ and M:WRITE for all reading and writing, files copied with the COPY routine will be treated according to the default conventions of the FORM, size, and BIN parameters of the !*COPY command. Deviation from inherent conventions is accomplished via FORM, size, and BIN parameter options.

For records being copied to the card punch, records containing a first byte of X'1C', X'3C', X'9F', X'BF', X'DF', X'FF', X'00', or X'78' are always punched in the binary mode; all other records are punched in EBCDIC. For all other devices, the distinction between binary and EBCDIC modes is meaningless because records are copied directly

without translation. Therefore, attempting to copy binary data to an EBCDIC device will result in meaningless output.

For paper tape, if BIN and size are not specified, the length of each binary record (first byte of X'1C', X'3C', X'9F', X'BF', X'DF', X'FF', X'00', or X'78') is always 120 bytes. When M:READ reads EBCDIC records from paper tape, it transmits only the number of bytes specified by the calling sequence to memory. Ordinarily, the COPY routine assumes that paper tape EBCDIC records have a byte count of 120. The size specification allows the user to override the standard count.

By assigning the X4 oplb to a RAD file or paper tape device before the !*OPLBS command is read, records copied from UI are adjusted to a 80- or 120-byte length, depending upon the contents of the first byte.

When copying or verifying a 9-track magnetic tape to a 7-track magnetic tape, UI and X4 should be assigned to the 9-track device.

If a record copied to the line printer or keyboard/printer contains more than 132 characters, only the first 132 are printed. Normally, the first character of the record is printed and single spacing is forced. Therefore, even if the first character is intended for format control, it will be printed as the first character of the print line in the normal mode. If the format option is specified, the first character is interpreted as a format control character and is not printed.

The BIN option should be used to copy nonstandard binary records. Paper tape codes NL, EOM, and ¢ are not interpreted as editing characters. All records are copies on a byte-for-byte basis. If paper tape is the input source, leading blanks are ignored and trailing blanks are included in the byte count. Paper tape !EOD NL is recognized as a file mark if it occupies the first five bytes of a record.

COPY OPERATIONAL LABELS

The following operational labels are used by the COPY routine in addition to the Utility subsystem operational labels:

Label	Usage
UI	Copy input.
X4	Verify input.
UO	Default copy output or second verify input.

Other operational labels may be used by COPY (at the option of the user) to specify a second input device for verifying or an output device for copying.

COPY OPERATING CHARACTERISTICS

The COPY routine checks whether input/output operational labels are assigned to the same physical device or same disk file as control input. If so, all control commands are read from the SI device and stored in memory prior to interpretation of the control commands to begin copying. When the SI and any input or output operational labels are assigned to the same physical device and attend mode is in effect, the message

```
** LD INPUT UI, ddnn
!!BEGIN WAIT
```

is written on the OC and DO device. The operator should load the input at this point and enter an S key-in to initiate the actual copy procedure.

If the operational labels are not assigned to the same physical devices, interpretation of control commands takes place as they are read from SI, and copying begins immediately without any message being output on the OC device.

CALLING COPY

The COPY routine is requested with the control command

```
!UTILITY COPY[,CORE]
```

where CORE specifies that, for the first !*COPY or !*VERIFY command, the records from the input device are stored in core in addition to being copied or verified. For subsequent !*COPY or !*VERIFY commands, these records in core, rather than those on the input device, are used as the input source. Following any !*COPY or !*VERIFY commands, record and file counts are displayed on the DO device.

After interpretation of the !UTILITY control command, control is transferred to the COPY routine which interprets the control commands listed below.

COPY CONTROL COMMANDS

OPLBS The !*OPLBS command identifies the operational labels of output devices to be used in COPY requests and input for comparison for VERIFY requests. The input for COPY operations is read from UI. For VERIFY operations, X4 is read. Operational labels may be assigned to any device. The form of the command is

```
!*OPLBS oplb1,...,oplbn
```

where oplb$_i$ ($i \leq 8$) is the operational label or fdun for an output device for subsequent !*COPY commands, or for an input device for subsequent !*VERIFY control commands. UI or X4, may not be specified. In the absence of an !*OPLBS command, the default is UO. (SI prestore mode is determined after each !*OPLBS command.)

COPY The !*COPY command causes records from the input device (UI) to be copied on the output device (specified in the !*OPLBS command) until the requested number of !EODs or file marks has been read and copied, or until the specified number of records has been copied. The form of the command is

```
!*COPY type[,number][,FORM][,size],[{BIN}][,P]
                                    [{ETA}]
                                    [{ATE}]
```

where

type is R if the number parameter refers to records, or F if the number parameter refers to files.

number has different meanings, depending upon the type parameter that precedes it. If the type parameter is R, "number" is the number of records to be copied, but refers to logical records for a blocked, sequential-access file. If "type" is F, "number" is the number of files to be copied, or is ALL, indicating that all files should be copied until two consecutive EOD images or file marks are copied. If "type" is F and any of the input/output devices is a sequential-access RAD file, "number" is 1 or it is omitted. If the number parameter is omitted, one record or file is copied.

FORM applies only if data is being copied onto the line printer or keyboard/printer. If the FORM parameter is omitted, single spacing of printed output is the format. If FORM is used, the first character of each record is used for format control and is not printed.

size specifies the maximum number of bytes in each record. If data is being copied to or from a sequential-access RAD file, "size" is the maximum logical record size and must be an even number. If "size" is omitted, all records are read and written in the standard record size (120 bytes). An !EOD card will not be recognized by M:WRITE if an odd byte count is specified or if a byte count of less than four bytes is specified.

BIN if omitted, mode (BIN or EBCDIC) is determined according to byte 1 of the record. If present, all copying is done in binary, either with the count specified in "size" or the standard record size (120 bytes) by default.

ETA specifies that the data read from the input operational label is to be converted from EBCDIC to ASCII before copying out to the devices specified on the !*OPLBS control command. Refer to Appendix E for the EBCDIC ⟷ ASCII translation table.

ATE specifies that the data read from the input operational label is to be converted from ASCII to EBCDIC before copying to the devices specified on the !*OPLBS control command. Refer to Appendix E for the EBCDIC ⟷ ASCII translation table.

P specifies that a page eject is to be performed on all !*OPLB line printer devices prior to and after execution of the current command.

BCOPY The !*BCOPY command causes records from a user blocked disk file or magnetic tape to be copied on the output device (specified on the !*OPLBS command) until the requested number of records or a complete file has been copied. This command allows for the case where the last record written is a short record by setting the SR flag on the write operation (see M:WRITE). The form of the command is:

```
!*BCOPY type[,number],,size[,{ETA}]
                             {ATE}
```

where type, number size, ETA, and ATE are defined as for the !*COPY command. Size must be specified equal to or greater than the actual record size to prevent loss of data.

VERIFY The !*VERIFY command requests comparison of data on the X4 device with data in core (CORE option) or with data from devices specified in the !*OPLBS control command. The form of the command is

```
!*VERIFY type[,number][,size][,{BIN}]
                               {ETA}
                               {ATE}
```

The parameters are defined as for the !*COPY control command.

Before the !*VERIFY control command is issued, it is assumed that all files have been repositioned, if necessary, by use of !*REWIND and other file positioning control commands (described in "Control Function Commands"). The entire verification process is completed when the number of files or records for verification has been compared.

DUMP ROUTINE

The DUMP routine is used to dump records or files onto an output device in either hexadecimal or EBCDIC format.

DUMP uses M:READ and M:WRITE for all input/output. If no mode is specified for dumping, all records are dumped according to the contents of the first byte of each record. Any record having a first byte of X'1C', X'3C', X'9F', X'BF', X'DF', X'FF', X'00', or X'78' is assumed to be a binary record containing 120 bytes, and is dumped with each data word being represented in EBCDIC as a 4-digit hexadecimal number. Any record that does not contain one of these characters in its first byte is assumed to be in EBCDIC and is dumped as such.

The user has the option to specify the byte count for paper tape record input, since M:READ pads all EBCDIC records with trailing blanks so that they appear to be fixed length in memory.

The HEX option for dumping should be used to dump non-standard binary records. The option causes all records that are to be dumped to be read in binary and dumped with each data word represented in EBCDIC as a four-character hexadecimal number. Since no editing is done when a binary read is specified, NL, EOM, and ¢ are not interpreted as editing characters. !EOD is recognized as a file mark.

DUMP OPERATIONAL LABELS

The DUMP routine uses the following operational labels:

Label	Explanation
LO	Output device for dumping.
UI	Input device for dumping, unless some other input device is specified.

DUMP OPERATING CHARACTERISTICS

If both SI and DUMP input are assigned to the same device, all of the control commands on the SI device are read and stored in memory before interpretation of the commands and dumping of the input tape begins. When this occurs, the message

```
** LD INPUT UI, ddnn
!!BEGIN WAIT
```

is written on the OC and DO device. The operator mounts the input tape and enters an S key-in to continue.

If SI and the tape device to be dumped are not assigned to the same device, no message is written and control commands are interpreted as they are read. The DUMP control commands are then listed on LL and dumping is performed.

CALLING DUMP

The DUMP routine is requested with the control command

```
!UTILITY DUMP[,oplb]
```

where oplb is the operational label or device input number of the input device. If the oplb is omitted or empty, the operational label is set to UI.

DUMP CONTROL COMMAND

DUMP The !*DUMP command causes records to be read from UI and written on the LO device in the specified mode until an !EOD or file mark is read, or the specified number of records has been read. The form of the command is

```
!*DUMP [number][,mode][,size]
```

where

number is a decimal integer. Only the specified number of records is dumped. If "number" is omitted, the file is dumped to an EOF or file mark. If "number" is ALL, the dump is performed to double file marks or !EODs.

mode indicates that all records on UI, regardless of the content of the first byte of each record, are written on the LO device in the mode specified. "Mode" is HEX for hexadecimal and EBCDIC for EBCDIC. If omitted, the record first byte sets mode.

size specifies the maximum number of bytes to be read in each record. If the dump "input" is a sequential-access RAD file, the size parameter must be an even number. For a blocked sequential-access file, "size" is the maximum logical record size. If it is omitted, the standard record size is used.

OBJECT MODULE EDITOR ROUTINE

The Object Module Editor is designed to maintain files containing sets of Xerox Standard Object Language modules. It generates or updates files by inserting and deleting object modules according to the program name in the start module item for each module. For each output file written, a list of module names is printed in the order of their appearance.

Object Module Editor is also used to list files containing object modules and to verify that the input object records contain no checksum or sequence errors.

A binary object module is defined as a sequence of binary records in Standard Binary format, each of which begins with a nonblank name item and terminates with a record whose first byte is X'9F' (END card) indicating that the record contains an end item.

A set consists of one or more object modules and is terminated by a file mark or !EOD. A tape may contain one or more sets and is terminated by double file marks or !EODs. Only one set of object modules can be contained in a sequential-access RAD file.

Note that the Object Module Editor routine does not maintain the object modules in the System Library and User Library areas on the RAD. These permanent areas are maintained via the RAD Editor (see Chapter 8).

OBJECT MODULE EDITOR OPERATIONAL LABELS

The Object Module Editor uses the following operational labels:

Label	Explanation
BI	Device-file from which binary object modules are to be inserted.
LO	Device-file for listing either UI or UO object module names.
UI	Input device-file.
UO	Output device-file.

OBJECT MODULE EDITOR OPERATING CHARACTERISTICS

Object Module Editor operates in two modes: list and modify.

In the list mode, only UI is read. The names of the object modules are printed on LO, and the checksum and sequence for each record are verified. After interpreting the !*LIST control command, the Editor checks if any two of SI[†], BI, and UI are assigned to the same device or disk file. If so, the message

```
** LD LIST UI, ddnn
!!BEGIN WAIT
```

is written on OC. The operator responds by preparing UI and entering an S key-in. Listing of the modules proceeds.

If no two of the labels SI[†], BI, or UI are assigned to the same device, control commands are interpreted as they are read and are written on DO. If the UI device is assigned to a sequential-access RAD file, the Object Module Editor leaves the list mode after reading the end-of-file.

In the modify mode, any modules to be inserted are read from the BI device and written on UO, as indicated by the control commands. If there are input files to be updated, they are read from UI. The names of all object modules written on UO are listed on LO. The object modules on BI must be in the same order in which they are to be inserted on UO, or BI must be rewound before each INSERT command. If BI is a disk file it is rewound with each INSERT command.

The Object Module Editor operates in the "prestore" mode (reading and storing commands before interpreting) when the conditions shown below occur; otherwise, the Editor operates dynamically.

Operational Labels[†] Assigned to Same Device-File	Prestored Data
SI, BI	SI
SI, UI	SI
BI, UI	BI
SI, BI, UI	SI, BI

BI is never prestored if assigned to a disk file.

After entering the modify mode, the Object Module Editor operates as follows:

If any two of the operational labels SI[†], BI, and UI are assigned to the same device-file. Object Module Editor follows the steps below:

1. Interpretation of control commands begins. If any object modules are to be inserted, and if SI and BI are assigned to the same device, the SI device is read until an !EOD is encountered and the message

   ```
   ** LD INSERTS UI, ddnn
   !!BEGIN WAIT
   ```

   is written on OC and DO. The operator loads the modules to be inserted on the BI device and keys in an S. If SI and BI are assigned to different devices or files, no message is written. The Editor then reads in all the modules on BI until either an !EOD or any other record with a first byte different from X'FF' or X'9F' is read from BI. Blank records are ignored.

2. If there are input files to be updated, the message

   ```
   ** LD INPUT UI, ddnn
   !!BEGIN WAIT
   ```

   is written on OC and DO. The operator must prepare UI and enter an S key-in.

3. The mode modification control commands are interpreted, causing updating or generation to proceed. Each control command is listed on LL as it is interpreted.

[†]Substitute OC for SI if a KP key-in is in effect.

If no two of the operational labels SI[†], BI, and UI are assigned to the same device-file, control commands from SI are read and interpreted dynamically. Records are read from BI and UI and written on UO in response to each mode modification control command. Every control command is listed on LL.

Object Module Editor uses M:READ and M:WRITE to perform all input/output. Each object module is identified by the program name stored in the start module item. No modules with blank names are even written on the UO tape.

CALLING OBJECT MODULE EDITOR

The Object Module Editor is requested with the control command

```
!UTILITY OMEDIT
```

After interpretation of the !UTILITY control command, control is transferred to the Object Module Editor routine. The control command and options available to OMEDIT are described below.

Object Module Editor begins reading control commands until an !EOD or an !*END is read, which terminates the input.

OBJECT MODULE EDITOR CONTROL COMMANDS

LIST The !*LIST command causes the Editor to enter the list mode. The names of the object modules on UI are read and listed on LO. Any checksum errors detected cause error messages to be written on LO, but listing continues. If the record is an !EOD, it is listed. If two consecutive !EODs are encountered, the Editor leaves the list mode and the next control command is interpreted. The form of the command is

```
!*LIST
```

[†]Substitute OC for SI if a KP key-in is in effect.

MODIFY The !*MODIFY command indicates that object modules are to be output on the UO device and causes the Editor to enter the modify mode. The modify mode terminates when an !EOD command is interpreted from SI. The form of the command is

```
!*MODIFY[{GEN   }]
        [{INSERT}]
```

where

GEN is an optional parameter indicating that object modules are to be selectively input from BI and that files are to be generated on UO. UI is not read. The control command !*MODIFY GEN may be followed only by !*INSERT control commands (GEN implies !*INSERT) used to define the elements to be selectively copied from BI to UO. No !*DELETE control commands may be used in the GEN mode.

INSERT must be specified if insertions from BI are to be read. If BI and UI are assigned to the same non-disk device, the complete BI file (up to an !EOD) will be prestored. Modules can be selected from BI by names on the !*INSERT control commands. The inserts must be in proper order. This command is used to update (input both !*INSERT and !*DELETE commands) UI and to write UO.

Note: If INSERT and GEN are omitted from the !*MODIFY control command, only !*DELETE control commands may be input.

MODIFY SYSTEM RBM SYSGEN magnetic tapes or any object module file can be rapidly and easily updated by use of !*MODIFY SYSTEM, a UTILITY OMEDIT control command. This command updates UI to UO with new object modules inserted from BI. Deletion and insertion are done in the sequence; read BI for IDENT, record back BI, delete UI object module with IDENT corresponding to that just read from BI, and insert new object module from BI. This process continues until the specified number of files are updated and written to UO. BI is rewound with this command. UI may contain mixed EBCDIC (80 byte) and standard RBM binary (120 byte) records.

Note: The first six records of the RBM SYSGEN system are nonstandard binary records and are copied automatically.

The form of the UTILITY OMEDIT control command is:

```
!*MODIFY SYSTEM,n
```

Any number of files (n) may be copied to UO from UI with binary object modules inserted from BI modules replacing those containing the same IDENT from UI. All BI object

modules are inserted until an EOF is encountered. UI is copied to UO until n files (default = 31) are written to UO. An additional EOF is then written to UO and return is to RBM JCP. No !*END or !EOD control command is required. (Note that !*END control command will cause a monitor CC abort.) BI must be assigned to a RAD or magnetic tape file with an EOF terminating the object modules. Object modules from BI must be in the order that they are encountered on UI. Appropriate error messages followed by a UT background abort results when errors are detected.

INSERT The !*INSERT command causes an object module to be inserted and is effective only in the modify mode. The form of the command is

```
!*INSERT name1[,name2]
```

where

$name_1$ is the name (up to eight EBCDIC characters) of the object module to be inserted.

$name_2$ is the name (up to eight EBCDIC characters) of the object module on UI that the $name_1$ object module must follow. If $name_2$ is omitted, the $name_1$ module is written following the module previously written on UO.

Modules to be inserted from BI must be in the same order as in the INSERT control commands. If GEN is specified on the MODIFY command, only the $name_1$ parameter in the INSERT command is required; if a $name_2$ is specified, it is ignored. If BI is a disk file, it is rewound with each INSERT command.

DELETE The DELETE command causes object modules to be deleted and is effective only in the modify mode. The form of the command is

```
!*DELETE name1[,name2]
```

where

$name_1$ is the program name (up to eight EBCDIC characters) of the first or only module on UI to be deleted.

$name_2$ is the program name (up to eight EBCDIC characters) of the last module on UI to be deleted. If absent, only one module is deleted.

The !*DELETE control command must name modules in the same order as their occurrence on UI.

RECORD EDITOR ROUTINE

The Record Editor is used for source editing by record number from any sequential device to any other sequential device. Record Editor provides the following capabilities:

1. Generates files containing source data.
2. Lists files containing source images in addition to associated line numbers.
3. Lists selected records in a file.
4. Modifies files containing source images.

RECORD EDITOR OPERATIONAL LABELS

The following operational labels must be assigned in addition to the standard Utility program operational labels:

Label	Explanation
SI	Input device for control commands.
LO	Output device for listing source images.
UI	Input device.
UO	Output device.

Note: Substitute OC for SI if KP key-in is in effect.

RECORD EDITOR OPERATING CHARACTERISTICS

The Record Editor routine operates in two modes: list and modify.

In the list mode, the Editor reads source images from UI and lists them on the LO device. It associates each image with a decimal line number, starting with 1.

In the modify mode, the Editor either updates or generates files on the UO device.

Record Editor uses M:READ and M:WRITE to perform all input/output. Therefore, all the paper tape editing and keyboard/printer editing that is standard to these routines is performed.

CALLING RECORD EDITOR

The Record Editor is requested with the following control command

```
!UTILITY RECEDIT
```

After interpretation of the !UTILITY control command, control is transferred to Record Editor, which begins reading control commands.

RECORD EDITOR CONTROL COMMANDS

A command requesting either the list or modify mode must immediately follow the !UTILITY command. All other control commands are interpreted as subcommands under each mode.

If a binary record is read from UI, Utility aborts after issuing the following message on OC and DO:

```
** MODE ERR UI, device
```

LIST The !*LIST command (list mode) causes the previous mode to terminate. The source files are read from UI and listed on LO. Each EBCDIC source image is listed along with an associated line number up to and including the first !EOD source image or file mark read. After the required number of files has been listed, another control command is read. Each !*LIST control command, file mark, or !EOD causes the line numbering to restart with 1. The form of the command is

```
!*LIST [number]
```

where number indicates the number of files to list. Listing continues until two consecutive !EODs are encountered or the specified number of files is listed. If "number" is omitted, one file is listed. If number is zero, the from/to parameters form limit pairs that define inclusively the records to be listed from the current file. Limit pairs must be in ascending order, except that two equal pairs cause only one record to be listed.

A !*MODIFY, !*END, or !EOD control command causes the list mode to terminate.

MODIFY The !*MODIFY command informs the Record Editor that files are to be either generated or updated. It terminates the previous mode and initiates the modify mode. The form of the command is

```
!*MODIFY [LIST][, GEN]
```

where

LIST indicates that a listing of records deleted or inserted will be produced on LO. If LIST is the only parameter used, the listing will contain the UI line numbers (the number deleted or the number preceding the one inserted). If GEN is also present, the UO line numbers will be listed.

GEN indicates that records are to be read from SI (there is no input on UI) and written on UO. If updating is to be performed (that is, there is input to be read from UI), the parameter field is left empty.

The modify mode is terminated whenever a !*LIST, !*MODIFY, !*END, or !EOD control command is input from SI. When the modify mode is terminated and GEN is specified, an !EOD or file mark is written on UO. When the modify mode is terminated and GEN is not specified, the remaining source images of the file on UI (until an EOD is encountered) are written on UO, followed by an EOD or file mark.

DELETE The !*DELETE command causes the indicated record source images to be deleted and is effective only in the modify mode. The form of the command is

```
!*DELETE number1[, number2]
```

where

$number_1$ is the line number of the first (or only) source image to be deleted.

$number_2$ is the line number of the last source image to be deleted.

INSERT The !*INSERT command causes record source images from SI to be added to UI and written onto UO, and is effective only in the modify mode. The form of the command is

```
!*INSERT number
```

where number is the line number that the insertions are to follow. If a line number of 0 (zero) is used, the insertions will precede the first line.

Every source image on SI following the !*INSERT control command is inserted until a new Record Editor control command is encountered.

CHANGE The !*CHANGE command causes the indicated source images to be deleted, and the source images following the CHANGE command to be written on UO. The command is effective only in the modify mode. The form of the command is

```
!*CHANGE number1[, number2]
```

where

$number_1$ is the line number of the first source image to be deleted.

$number_2$ is the line number of the last source image to be deleted. If omitted, only one source image will be deleted.

Following the !*CHANGE control command, every source image on SI is inserted until another Record Editor control command is encountered.

SEQUENCE EDITOR ROUTINE

The Sequence Editor edits EBCDIC card images by sequence number. It is more flexible than the Record Editor in that multiple programs or sections of programs may be updated and sequenced individually within single or multiple files. It provides greater protection from updating in an incorrect sequence, or from accidentally updating the wrong program. Another feature of the Sequence Editor routine is that update card images may be inserted without changing the existing sequence numbers. Thus, update decks may be cumulative and will reflect the development of a source program.

The Sequence Editor is primarily intended for installations where EBCDIC source programs are kept on magnetic tape. It is somewhat impractical for paper-tape-oriented systems or systems without a line printer.

Editing is accomplished by designating columns 73 through 80 of a source card image as the "sequence field". This field consists of two parts, the ident and the sequence number.

The optional ident is that portion of the sequence field that uniquely identifies a program or program segment. If defined, the ident begins in column 73 of the card image and is from one to six alphanumeric characters in length.

The required sequence number is that portion of the sequence field that is sequenced numerically. It consists of from two through eight decimal characters and ends in column 80 of the card image. The user can specify the value by which successive sequence numbers are incremented. In general, a large sequence increment will allow larger insertions without affecting the existing sequence numbers.

Together, the ident and sequence number must not total more than eight characters. Any unused columns will be between the ident and the sequence number and will be ignored by the Sequence Editor.

SEQUENCE EDITOR OPERATIONAL LABELS

The following operational labels are used by the Sequence Editor routine:

Label	Explanation
SI	Update data (includes card images and control commands). Not effected by KP key-in.
LO	Annotated listing of added and deleted card images.
UI	Input device.
UO	Output device.

Device, above, refers to any permanent storage device such as magnetic tape, paper tape, or RAD (single sequential file). Note that LO should not be assigned to the keyboard/printer, because the sequence number portion of the printout is truncated on that device.

SEQUENCE EDITOR OPERATING CHARACTERISTICS

The Sequence Editor performs two separate and distinct functions: generates files on UO from source images input on UI and updates files from UI onto UO, taking updates from SI. Only one of these functions can be performed per call to the Sequence Editor (SEQEDIT).

The file generation (GEN) function is used to create the permanent files initially. It is required that files be sequenced as they are generated. The user can generate one file (terminated by a file mark) wherein a single file mark is written on UO, or multiple files (terminated by two file marks) wherein two file marks are written onto UO and UO is backspaced one file.

The update function is used to update UI by replacing, deleting, or inserting card images from SI and writing the updated files onto UO. The files can be resequenced as they are written. The user can update one file (terminated by an EOF from UI) wherein an EOF is written onto UO, or all files (terminated by logical end-of-tape or two EOFs from UI) wherein two file marks are written on UO and UO is backspaced one file. With the ALL option, it is not necessary to update each file, but all files will be copied onto UO.

Sequencing is a separate operation in that the card images are sequenced as they are written on UO. Thus, it is possible to update an existing file by ident and sequence number while placing a new ident and sequence number on the update file.

CALLING SEQUENCE EDITOR

The Sequence Editor is requested via the control command

```
!UTILITY SEQEDIT,[GEN][,IGN][,ALL]
```

where

GEN indicates that output files are being generated on the UO device and that there are no input files to be updated.

IGN in update mode indicates that UI sequence errors are to be ignored if UI is being updated. If IGN is used, no sequence error messages are printed.

In GEN mode, IGN indicates that UO is not listed.

ALL indicates that the function is to continue until two EOFs are encountered from UI.

The leading comma must be specified.

The Program Executive transfers control to the Sequence Editor, which interprets and validates the parameters. If illegal parameters are input, the Utility program aborts with a code of UT. If this is an update (the GEN option was not specified), the following message is output on OC and DO.

```
** LD INPUT UI, ddnn
!!BEGIN WAIT  (if attend mode only)
```

SEQUENCE EDITOR GENERATE CONTROL COMMAND

SEQUENCE The !*SEQUENCE command is used to sequence columns 73 through 80 of the card images on UO. Only one file can be sequenced with each !*SEQUENCE command. The form of the command is

```
!*SEQUENCE sequence field,increment
```

where

sequence field contains the sequence number of the first sequenced card image to be written on the output tape.

increment is the sequencing increment number. If omitted, an increment of 10 is used. It is the responsibility of the user to ensure that the sequence number does not get incremented past the size of the sequence number field. No warning is issued if this overlap occurs.

SEQUENCE EDITOR UPDATE CONTROL COMMANDS

IDENT The !*IDENT command defines the breakdown of the sequence field into the ident and the sequence number. It applies to card images from UI and SI only. If used, it should precede the update cards to which it applies. If omitted, the ident field is considered empty and the sequence number is eight characters in length. The !*IDENT control command is used whenever it is necessary for the Sequence Editor to know the size and content of the ident field (that is, when UI contains multiprogram files or single-program files with nondecimal characters in the sequence field). It is not to be used when files are being generated. The form of the command is

```
!*IDENT [ident][,sequence-number]
```

where

ident is an integer n_1 ($0 \leq n_1 \leq 6$) that specifies the number of characters in the ident subset of the sequence field starting from column 73. If "ident" is omitted, the ident field does not exist.

sequence-number is an integer n_2 ($2 \leq n_2 \leq 8$) that specifies the number of characters in the sequence number subset of the sequence field ending in column 80. If omitted, sequence number is set equal to the difference (8 - ident).

The user should note that if a nonzero ident field has been specified on an !*IDENT command, the idents on each card image from UI must match exactly or resequencing will be suspended when the first nonmatching ident is encountered. Hence, if UI is known to have nonmatching idents (for example, a file that has never been sequenced or one that has been updated and contains some blank sequence fields), a separate sequence operation should be performed (without a simultaneous update) specifying an empty ident field.

Replacement. The update card itself, rather than a control command, is used to replace a card image from UI. The sequence number on the update card must equal the sequence number on the UI card image to be replaced. The card image for UI and the message "DELETED", followed by the card image from SI and the message "INSERTED" are output on LO.

Insert. The update card itself, rather than a control command, is used to insert a card image on UO. The sequence number on the update card must be between the sequence number of the two continuous UI card images where the update card is to be inserted. The card image from SI and the message "INSERTED" are output on LO. Cards without sequence numbers are inserted immediately following the sequenced card preceding them. Thus, a large block of card images can be inserted by placing the proper sequence number on the first card only. The nonsequenced cards will be written on the output tape without sequence numbers. It is recommended that the tape be resequenced as it is being updated if unsequenced cards are inserted.

DELETE The !*DELETE command deletes one or more card images from UI. Nonsequenced cards can only be deleted by deleting from the last sequenced card preceding the nonsequenced card(s) up to and including the next sequenced card. Deleted card images are listed on LO. The form of the command is

```
                                                    73          80
!*DELETE [sequence field2]                          sequence field1
```

where

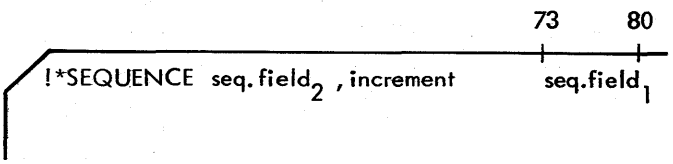
sequence $field_2$ indicates that the images are to be deleted from the ident and/or sequence number in sequence $field_1$ up to and including the ident and/or sequence number in sequence $field_2$.

sequence $field_1$ contains the ident and/or sequence number of the first or only card image to be deleted from UI. This parameter is required.

SUPPRESS The !*SUPPRESS command is identical to the !*DELETE control command except that no deletion and images are listed on LO. The form of the command

```
                                                    73          80
!*SUPPRESS [sequence field2]                        sequence field1
```

SEQUENCE The !*SEQUENCE command is used to resequence columns 73 through 80 of the card images on UO. Only one program can be resequenced with each !*SEQUENCE command. Therefore, resequencing is suspended when either a file mark or a card image with a sequence number identifying a new program is written on the output tape. Resequencing is also suspended when another !*SEQUENCE command is executed; therefore, parts of a program as well as entire programs can be resequenced. The form of the command is



where

sequence field$_2$ contains the ident and/or sequence number of the first resequenced card image to be written on the output tape and does not necessarily have the same fields as defined in the !*IDENT command. (The !*IDENT command defines sequence fields for the input tape and update data only.) If omitted, resequencing is suspended.

increment is the resequencing increment number. If omitted, an increment of 10 is used. It is the responsibility of the user to ensure that the sequence number does not get incremented past the size of the sequence number field. No warning is issued if this overlap occurs.

sequence field$_1$ contains the ident and/or sequence number from UI at which the !*SEQUENCE command becomes effective. If omitted, the !*SEQUENCE next card image to be written on UO.

UTILITY ERROR MESSAGES

Table D-5 of Appendix D lists the error messages issued by the Utility Subsystem. Unless otherwise noted, the following definitions apply in these messages:

Code	Explanation
oplb	Operational label of the device.
device	Device type or physical device number.

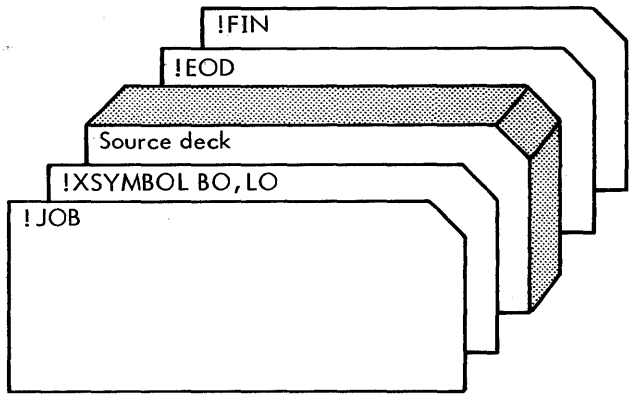
The operator response to a !!BEGIN WAIT message on OC may be any valid, appropriate, RBM unsolicited key-in, such as S to continue processing, or X to abort job. Other appropriate key-in may precede an S key-in if desired. The !!BEGIN WAIT message is used only if attend mode is in effect.

10. 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.

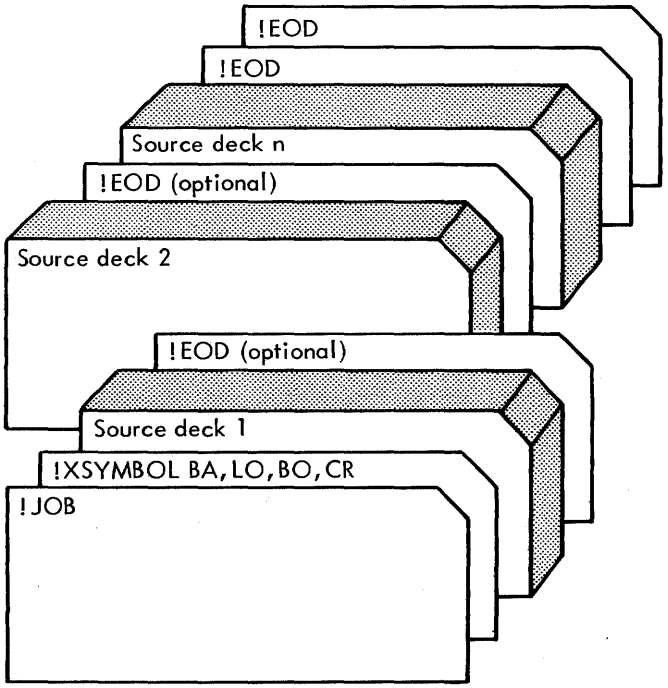
EXTENDED SYMBOL EXAMPLES

ASSEMBLE SOURCE PROGRAM, LISTING OUTPUT AND BINARY OUTPUT



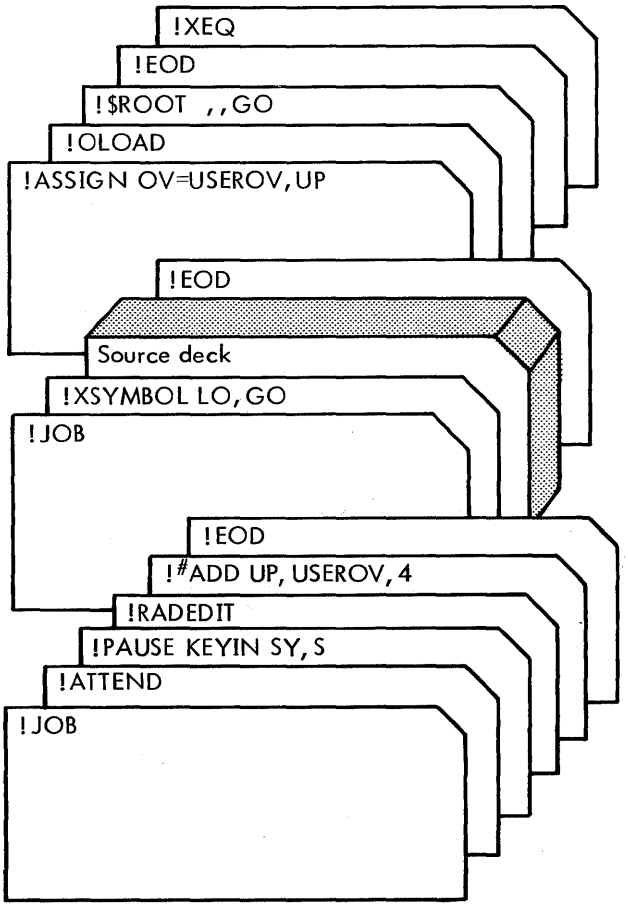
In this example, the symbolic input is received from the SI device (always defaulted), the binary output is received on the BO device, and the listed output is received on the LO device. Note that although BO and LO are normally default cases, they must be specified if output to the GO file (also a default) is not desired.

ASSEMBLE IN BATCH MODE, LISTING OUTPUT AND BINARY OUTPUT WITH SYMBOL CROSS-REFERENCE



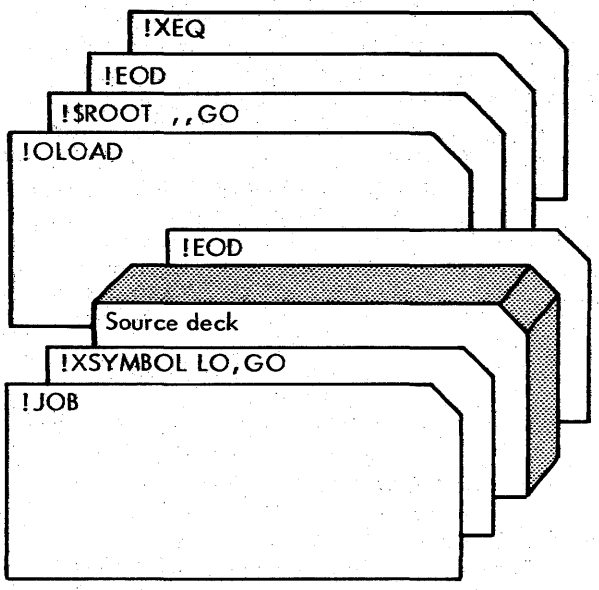
In this example, the source decks are assembled in batch mode (BA). In this mode, successive assemblies may be performed with a single !XSYMBOL command until a double !EOD command is encountered. The parameters defined on the !XSYMBOL command will hold true for each assembly in the batch. Each assembly will be followed by a Symbol cross-reference (CR).

ASSEMBLE, LOAD, AND GO FROM USER DEFINED OV FILE, LISTING OUTPUT



In this example, the user is defining his own OV file through a call to the RAD Editor. After assembly, the OV file is assigned to the user defined file. The call to the Overlay Loader (!OLOAD) causes it to load the module defined on the !$ROOT command to the USEROV file for execution. The advantage to assigning the program to a user-defined OV file rather than using the RBMOV file is that the program can be loaded into core for execution repeatedly without reassembly. Conversely, the contents of RBMOV cannot be guaranteed to be saved from one job to another.

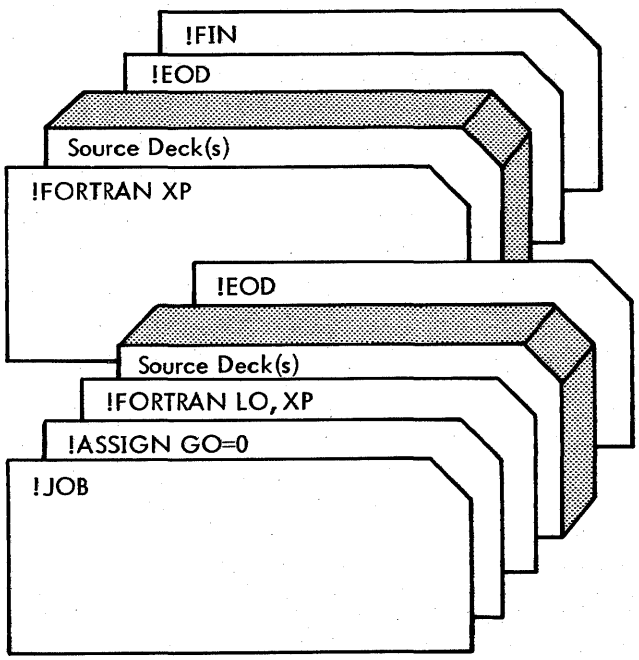
ASSEMBLE SOURCE PROGRAM, LISTING OUTPUT, LOAD AND GO



In this example, the binary object module is loaded into the RBMGO file located in the System Data area. The call to the Overlay Loader (!OLOAD) causes it to load the module defined on the !$ROOT command to the RBMOV file for execution. The double comma on the !$ROOT command informs the Loader that the temp, exloc parameter options are defaulted.

BASIC FORTRAN IV EXAMPLES

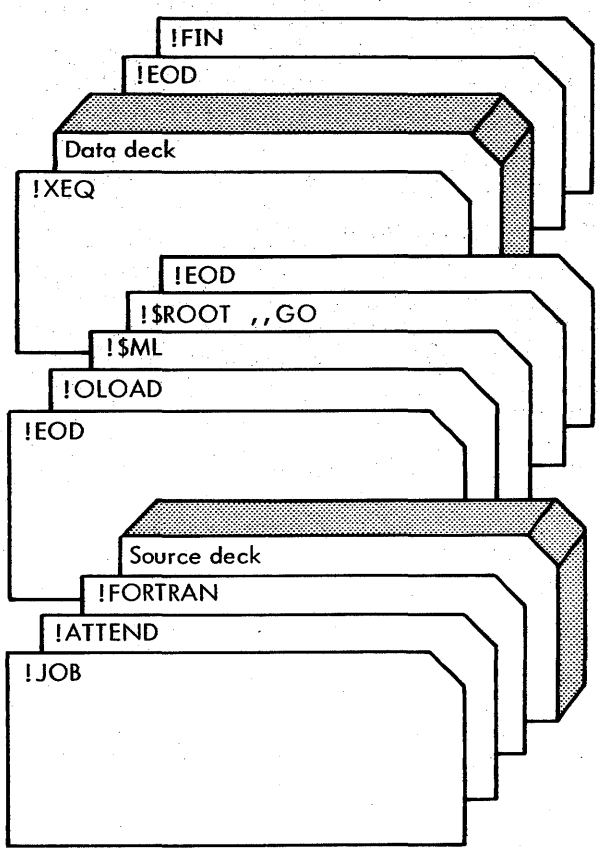
COMPILE MULTIPLE PROGRAMS



In this example, output to the GO file is not desired in the first job, so the GO oplb must be assigned to 0 (see Appendix E and !ASSIGN command writeup in Chapter 2). An object listing is desired (LO) and extended precision real data is specified.

The second job will receive a source listing by default and extended precision real data is again specified. Since the parameters are different on the two !FORTRAN control commands, the jobs cannot be run in batch mode.

COMPILE, LISTING OUTPUT, LOAD AND GO



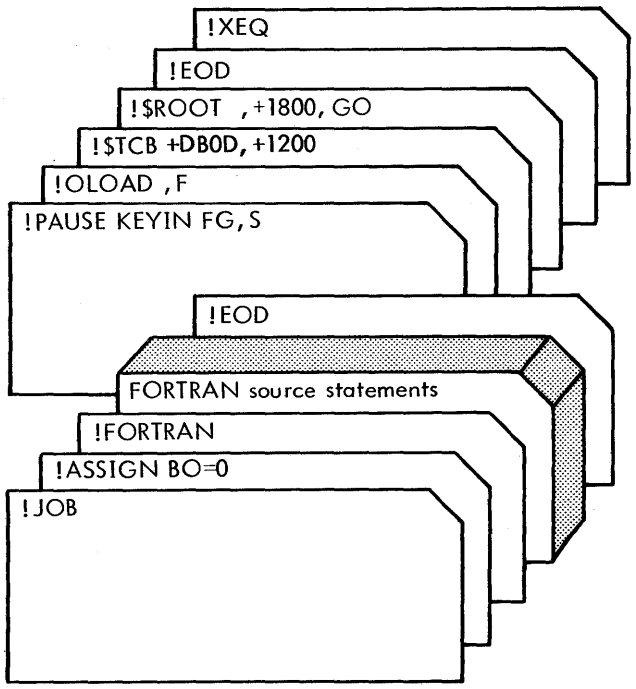
In this example, the !ATTEND command specifies that the Monitor is to go into a "wait" state instead of aborting the job in case of irrecoverable error (generally recommended for "load and go" jobs). Binary output will be received on both the BO and GO devices by default, and standard precision mode is also assumed by default. The binary object module is loaded into the RBMGO file located in the System Data area.

The call to Overlay Loader (!OLOAD) causes it to load the module defined on the !$ROOT command to the RBMOV file for execution. The double comma on the !$ROOT command informs the Loader that the temp, exloc parameter options are defaulted. The Loader is

requested to output a LONG map (!$ML). The !XEQ command causes the executable program to process the data deck.

COMPILE AND EXECUTE FOREGROUND PROGRAM

This example would be used for debugging purposes only.

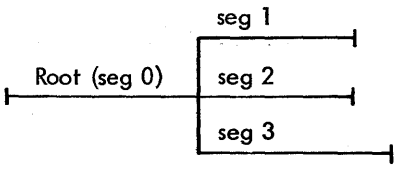


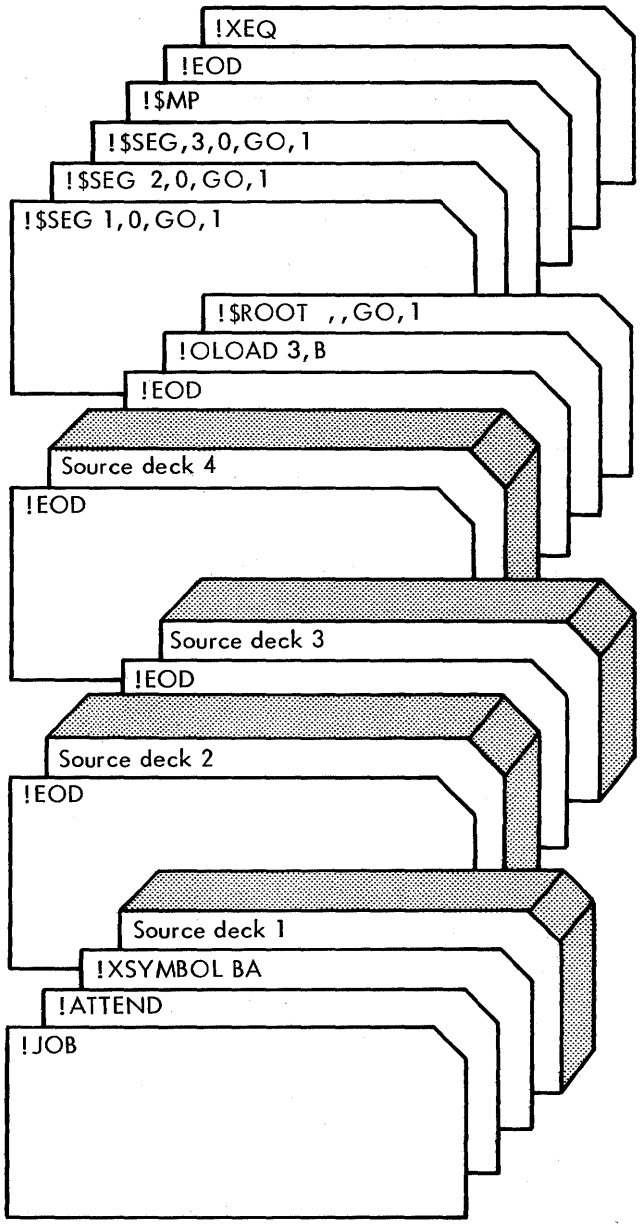
In this example, binary output to the BO device is suppressed. The !FORTRAN control command specifies that the binary output is to be received on the GO file by default and standard precision mode is assumed. The !PAUSE command permits the operator to key in FG, S to access protected foreground memory. The program is defined to the Overlay Loader as a foreground program (!OLOAD, F) and the COMMON base is set to the FWA of the background. The Loader is to create the Task Control Block, the first two words of which are defined on the !$TCB command. These two words specify that the task is to be connected to interrupt location X'10D' (Intergral interrupt number 2, priority level 8, within group 0).

The !$ROOT command specifies that the root is to be loaded from the GO file, and will start execution at location 1800 in foreground memory. The core image form of the program is loaded on the OV file (RBMOV). The !XEQ command loads the executable program into core. When loaded, the task is armed, enabled, and then triggered.

SEGMENTED PROGRAM EXAMPLES

ASSEMBLE SEGMENTED BACKGROUND PROGRAM, LOAD AND GO

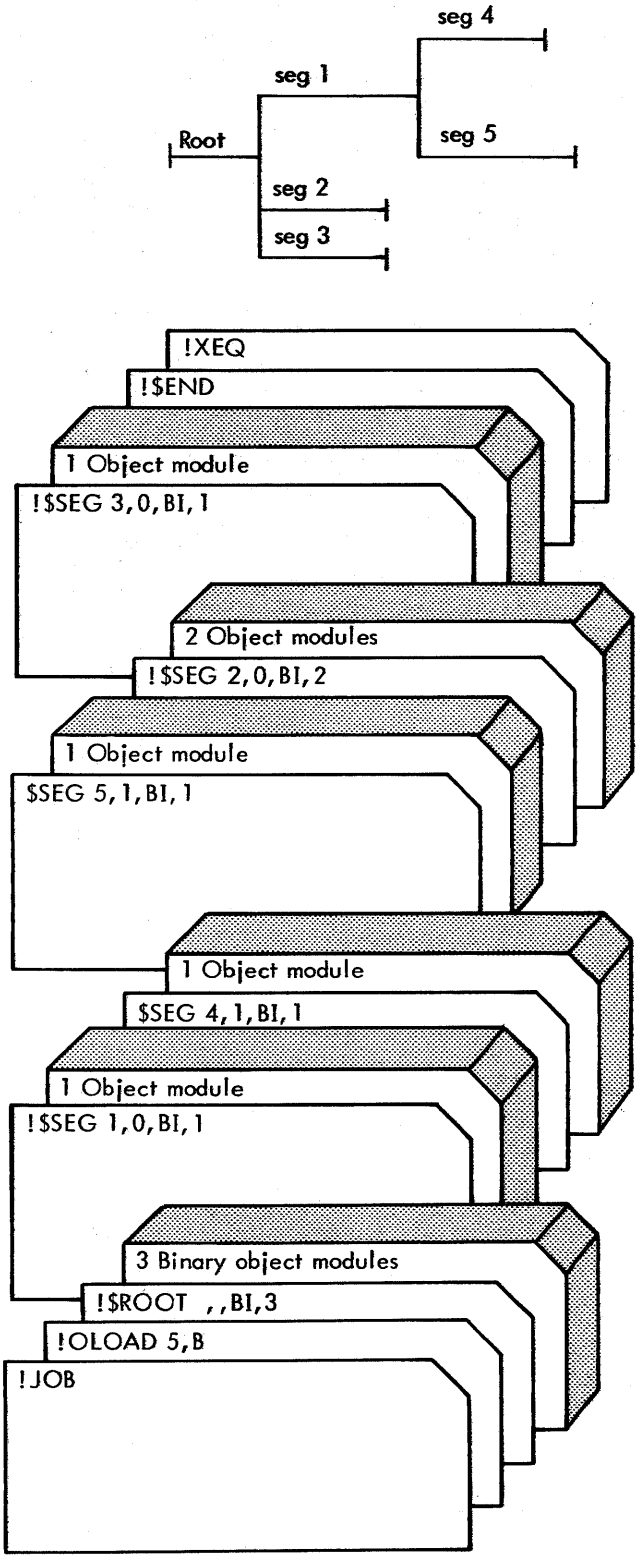




Given the program tree structure shown above, the sample deck setup illustrates a background program with a root and three overlay segments. These are assembled and loaded into the RBMGO file. The !OLOAD command specifies that these three segments are to be loaded, and defines it

as a background program (B). The $SEG commands specify that segments 1 through 3 are attached to the root, and the modules are to be loaded from the RBMGO file to the RBMOV file for subsequent loading into core for execution. A load map is output (!$MP).

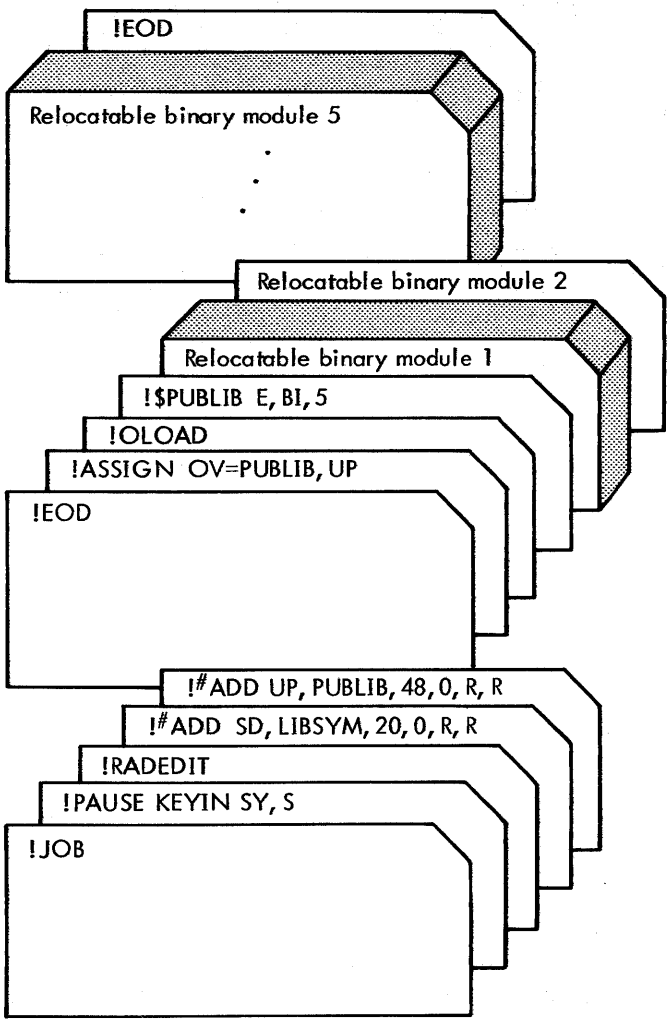
LOAD AND EXECUTE MULTIPLE OBJECT MODULES



Given the sample program tree structure shown above, the illustrated deck would load and execute the segmented program. The program is loaded from either the device or file assigned to the BI operational label. No load map is requested (an !$ML, !$MS, or !$MP command could be inserted after the !OLOAD command if a map was desired). Although the segments could be loaded in any order, the proper calling sequence is the responsibility of the user.

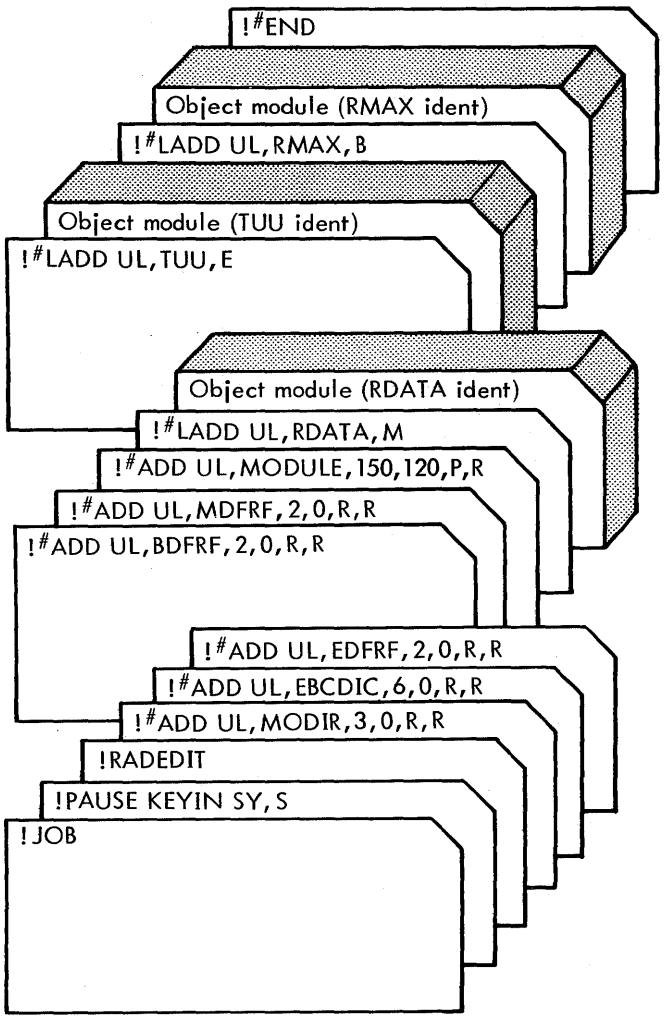
RAD EDITOR EXAMPLES

BUILD PUBLIC LIBRARY



The Public Library is core resident. In this example, the user must create two RAD files to set up the Public Library: the LIBSYM file and the PUBLIB file. The LIBSYM file contains the Symbol Table for the Public Library and is used by the Overlay Loader to satisfy references to the Public Library. The PUBLIB file contains the Public Library and is booted in with RBM. (RBM must be rebooted to load the updated Public Library.)

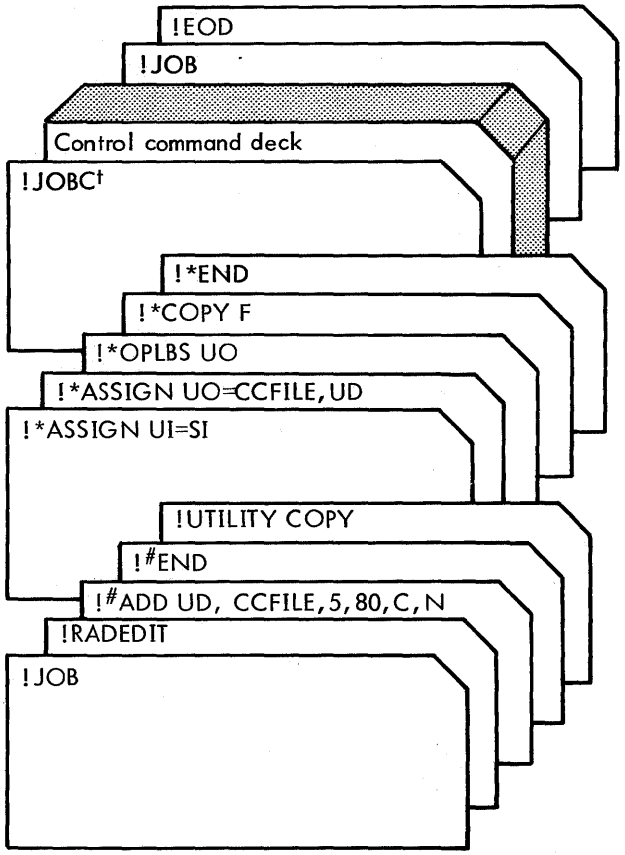
LOAD ROUTINES IN USER LIBRARY



In this example, the User Library requires the following six files to be allocated in the User Library area (UL): MODIR, EBCDIC, EDFRF, BDFRF, MDFRF, and MODULE. The !#LADD command enters the routines into the defined four files, depending on the library code parameter on the !#LADD command: Basic (B), Main (M), or Extended (E). The same basic method is used to set up the System Library.

UTILITY EXAMPLE

CREATE A CONTROL COMMAND FILE



In this example, the job stream will create the compressed file CCFILE in the User Data area. Control commands will be read from the SI device into file CCFILE. The job stream on CCFILE may now be executed by assigning CC = CCFILE,UD. Note that CCFILE must not have a !JOB command on its first entry, since this would immediately transfer CC back to the SYSGEN assignment. However, it is often convenient to end the control command file with a !JOB command to initiate a return to the SYSGEN assignment.

---

†A !JOB command must not be the first card in the control command deck; !JOBC is permissible.

11. SYSTEM STARTUP

The startup of an established RBM system (that is, subsequent to system-generation time) is normally performed either by "booting" from a self-loading 'system-save tape', or by booting directly from the system disk[†] if the latter has not been disturbed (e.g., used for another purpose) since the previous shutdown of the system. As part of this startup process, updating of the public library and the resident foreground can be achieved, as well as absolute system patching applied to both core and RAD images.

Other forms of initial system loading and system updating are described in the RBM/SM Reference Manual, 90 30 36.

SYSTEM SAVE TAPE

A system save tape is produced by assigning operational label B0 to magnetic tape and using the #SAVE function of the RAD Editor, omitting the FILE parameter, while the system is in an operational state (e.g., prior to shutting down). All RAD and/or disk pack areas necessary to subsequent system operation should be specified to be saved. The tape will contain a first block bootstrap routine, a restore program, and the saved disk areas. After the restoration of the system disk image, the restore program automatically initiates an RBM boot from the system disk device.

The restore program issues the following message:

```
RESTORING VERSION xx OF mm/dd/yy hrmn
```

As each area is restored, the message

```
RESTORING area TO dn     (dn = device number)
```

is issued, unless DATA switch 2 is up. If the area is the first area being restored to a disk pack or cartridge disk, the message

```
IDLE, RUN TO WRITE
```

is issued. If it is permissible to write on the indicated device, the operator must move the COMPUTE switch to IDLE and back to RUN in order to continue.

If an end-of-tape condition is sensed, the restore program will rewind the tape off-line (i.e., set it to the manual mode), output the message

```
MOUNT NEXT REEL
```

and attempt to read the original tape device number. The restore operation will continue when the next reel is mounted and the tape drive is placed in the automatic mode (by pressing START on the tape drive control panel).

Table 20 lists the error messages that may be output while restoring the system to the RAD/disk.

Table 20. Save-Tape Restore Error Messages

Message	Restore Program Action	Operator Action
WRITE PRO RAD	Keep trying to perform write operation.	Reset RAD write-protect switches.
SEQ ERR	Keep trying to read the tape successfully.	Restart or abort[†].
CHECK WRITE ERR	Keep trying to perform write/check-write operation.	Restart or abort[†].
CHSM ERR	Keep trying to read the tape successfully.	Restart or abort[†].
TAPE TRANS. ERROR	Keep trying to read the tape successfully.	Restart or abort[†].
RAD TRANS. ERROR	Keep trying to perform write operation.	Restart or abort[†].

[†]To restart, rewind tape and reboot; to abort, activate PCP interrupt which causes loading of the current area to be abandoned. Following the abort, the tape is searched for the next area to be restored.

RBM BOOT PROCEDURE

The RBM boot procedure is essentially the same whether the system is loaded directly from the system disk or from a self-loading save tape (of the form described above) via the disk. The principal difference is that in the former case a standard hardware-load operation is initiated from the system-disk device; in the latter case, from the tape drive on which the save tape is mounted (the subsequent load from disk is automatic). In either case the actual boot process is effected by the loading to memory and execution of the RBM bootstrap record.

The RBM bootstrap will initially move itself to high core and then read in RBM from the system processor area of the RAD. The information necessary to read in RBM is contained in

[†]RAD or disk pack.

the bootstrap and is supplied at system generation time when the bootstrap is written on the RAD. After the resident portion of RBM is loaded, control is transferred to another bootstrap that loads the remainder of the RAD. This second bootstrap functions in the overlay region of the RBM.

The second bootstrap initially inputs the Transfer Vector Table to complete the loading of the resident portion of RBM. Next, if DATA switch 4 is not set, an attempt is made to assign an operational label to the PUBLIB file in the user processor area. If a Public Library is present, the assignment will be made and the bootstrap then inputs the Public Library. If DATA switch 4 is set, the Public Library will not be loaded. After the Public Library is processed, a Hex Corrector patching routine (see below) will be activated if DATA switch 1 is set. If DATA switch 3 is not set, the bootstrap then searches the SP, UP, and FP area directories (in that order) for all files flagged as a resident foreground file. All such files are loaded one at a time as they are encountered in the file directory and their initialization routine is executed if one exists. The initialization routine can do any required housekeeping (such as repositioning all appropriate files), arm and enable the appropriate interrupts, and then return control to bootstrap. (The initialization routine is linked by an RCPYI P, L instruction.) It expects to have control returned to the address in the L register. Hence, the bootstrap will read in the resident foreground programs one by one and execute any initialization routine unless DATA switch 3 is set.

The system is then completely loaded and the bootstrap sets the protection registers, outputs the following messages (if DATA switch 2 is not set), and enters a wait state:

```
!!AFTER 'WAIT' SET PROTECT
!!SET PARITY
!!KEY-IN 'S' TO BEGIN
```

If the computer enters a "wait" state before the above messages are output, the bootstrap was not successful in loading the required data. This would usually be caused either by a parity error while reading the RAD or by a faulty foreground program.

Note that if the above messages are inhibited by setting DATA switch 2 prior to execution of the boot, the operations indicated by the messages should still be performed, however, in order to ensure system integrity.

PUBLIC LIBRARY CREATION OR UPDATING

The Public Library can be created and thereafter can be completely regenerated any time the user desires. A file with the name PUBLIB will have to be defined via the RAD Editor in the User Processor area for the Public Library, and a file named LIBSYM must be defined in the System Data area of the RAD. The relocatable binary decks of all routines to be specified as being in the Public Library are loaded by the Overlay Loader (via the !$PUBLIB control command) and an absolute core image version is written by the Overlay Loader on the RAD file defined as PUBLIB.

Before executing the Overlay Loader, the operator must key in SY so that the Loader can write in a protected RAD file.

When a Public Library is successfully loaded, additional updating of RAD files will be done by the Overlay Loader. The Public Library Transfer Vector Table will be input from the RAD and either created (for an initial load) or updated for succeeding loads. This process consists of linking each Public Library definition (DEF) in the Symbol Table to a transfer vector and linking the transfer vector to the value of the DEF. When the linkage is completed, the Overlay Loader writes the new Public Library Symbol Table into a previously defined file (called LIBSYM) in the system data area of RAD. For an initial load, this file will be previously defined, via the RAD Editor, with the name LIBSYM. The new Transfer Vector Table is then written on the RAD (replacing the previous one), and the Loader exits to M:TERM. (Note that RBM must be rebooted from the RAD in order to load the Public Library into core memory.) The Public Library should not be loaded into core (by rebooting the system from the RAD) until the user has reloaded all foreground and background routines that use the Public Library.

RESIDENT FOREGROUND CREATION OR UPDATING

Resident foreground program files must be defined via the RAD Editor. These files may be in the System Processor (SP), User Processor (UP), or Foreground Program (FP) area of the RAD. Also, the parameter (RF) on the !#ADD command specifying that this is a resident foreground file will have to be set. One RAD file can be defined for each foreground program, thus allowing an update to be done on a program basis as opposed to the entire resident foreground area. The Overlay Loader reads in a relocatable binary deck of each foreground program and creates an absolute core image version of the program in its predefined RAD file. Foreground programs assembled as absolute sections must be loaded with an ABS control command. Prior to executing the Overlay Loader, the user may key in SY to specify that the protected RAD files can be written on.

For an update, only those programs being modified need be reloaded. However, if a program exceeds its allocated core space, other programs must be reloaded and relocated at a new absolute address in a different area of core.

The Overlay Loader (or the Absolute Loader) will store in the first sector of each file the appropriate header information that the RBM bootstrap needs to load and initialize each foreground program. The information needed by the bootstrap consists of the following items:

1. Load address.
2. Number of bytes in program.
3. Entry address of initialization routine (if present).

If no initialization routine is specified, the RBM bootstrap will initialize the task's interrupt level from information in

the TCB. The task may also be triggered at this point if the TCB so specifies.

After a resident foreground program is created on the RAD and is flagged for automatic boot-time loading in its file-directory entry, it is brought into core by manually rebooting the system from the RAD. It can also be brought into memory by inputting a !processor or !XEQ command with oplabel OV assigned to its RAD file.

SYSTEM PATCHING

Patches to the Monitor or Public Library may be loaded at boot time if DATA switch 1 is set. Monitor patches will also be written to the RAD, thus ensuring a permanent change to all future boots. All patch cards have the form

$$aaaa,cccc_1\ [,cccc_2,cccc_n][*comments]$$

where

aaaa is the first (or only) absolute core memory location to be modified.

$cccc_i$ are the desired (hexadecimal) contents of aaaa and the following n-1 locations.

Patches may also be loaded dynamically to user program (or the Monitor) in either of two ways.

1. Following a HEX control command.

2. Following an unconditional H key-in.

All patch decks are terminated by an EOD control command. To patch relative to the start of program modules, a bias card may be used. Its form is

$$+\begin{Bmatrix} bbbb \\ ID\begin{Bmatrix} PA \\ xx \end{Bmatrix} \end{Bmatrix}$$

where

bbbb is the bias (load origin of the program) and the following correctors are loaded relative to that location.

PA means that the following patches are to be loaded relative to the RBM Patch Area.

xx is an RBM overlay ID; thus the corrections following the bias card are loaded relative to the overlay base.

Note: All patches at boot time to the monitor or a monitor overlay will be written to the RAD. At other times, three cells of the RBM Patch Area are needed for each overlay patch. The overlay length is also expanded to the next sector boundary (or maximum of 512 words) to allow use of the end of the overlay as a dynamic patch area.

Any value on a patch card preceded by an R ($Rcccc_i$) will have the current bias added to it. Any value on a patch card preceded by a P ($Pcccc_i$) will have the bias of the RBM Patch Area added to it. Any value on a patch card preceded by an O ($Occcc_i$) will have the bias of the RBM overlay area added to it. Any value on a patch card preceded by a J ($Jcccc_i$) will have the bias of the JCP added to it.

The programmer must not modify the first and last cells of the Patch Area, as the first contains the length of the Patch Area and the last contains the number of temporary RBM overlay patches. As mentioned previously, three words of the Patch Area are needed for each overlay patch, taken from the top of the Patch Area downward. When an RBM overlay is read into core, the Patch Area is searched for patches for that overlay. If any are found, they are applied before control is passed to the overlay.

12. DEBUG

INTRODUCTION

This chapter describes the use of Debug and its interface with RBM.

GENERAL DESCRIPTION

The RBM Debug package is a debugging tool primarily designed for nonoverlaid background programs, with limited facility for foreground programs. It provides the user with the following capabilities:

1. To transfer control to the control device from a specified location in the user's program or through the Control Panel Interrupt.
2. To dump selected core and registers on the keyboard/printer or the line printer.
3. To modify memory locations and registers.
4. To logically insert code at specified memory locations.
5. To begin or continue execution at a specified memory location (i.e., selective execution).
6. To perform conditional memory dumps (snapshots) of registers and selected core locations at a specified location and optionally transfer control to the control device.
7. To step through a program.

FOREGROUND USER'S DEBUG CAPABILITY

Debug can be used to aid the checkout of a foreground program operating at priority levels lower than the Control Panel Interrupt level. To accomplish this, Debug is moved to the Control Panel Interrupt level, where it may be directly entered by pressing the Control Panel Interrupt switch. The Control Task remains at the lowest interrupt level. Key-ins requesting Control Task functions may be made by typing a "slash" (/) followed by NEW LINE (NL) in response to the DKEYIN message. Snapshots may be placed in all tasks whose interrupt level is lower than the Control Panel Task.

OVERLAY USER RESTRICTIONS

When a snapshot is inserted in a currently resident segment using a Debug control command, the snapshot is valid only until the segment is overlaid, since Debug operates only at execution time on resident programs. This problem is reduced by allowing the user to assemble Debug calls into his program.

RBM AND FOREGROUND USER'S INTERFACE

Debug is normally a subtask of the RBM Control Task with a priority just below the IDLE subtask. Debug is triggered by any of the three resident Monitor routines (D:SNAP, D:KEY, or D:CARD), by the KEYIN subtask, or by the Job Control Processor (JCP). JCP triggers Debug when it receives an XED command, and the system loader transfers control via D:KEY. When a foreground user wishes to use Debug, he gives control to Debug by an !XED card or by an unsolicited key-in of DE. After Debug has control, the foreground user moves Debug to the Control Panel Task level with a Define command. After debugging, the foreground user issues the Debug command Q which restores Debug to its original level.

MEMORY REQUIREMENT AND INSERTION BLOCK DEFINITION

The executive portion of Debug is a foreground program that may be resident or nonresident. If the program is resident, it must be so specified when the Debug file is created with the RAD Editor. It is read into core when RBM is booted. If the program is nonresident, it is loaded like any other foreground program (see Chapter 6). Debug has the following core memory requirements:

1.	Executive	440 locations
2.	Zero table	35 locations
3.	Overlays	RBM overlay space
4.	Insertion block	User-defined

The insertion block is an area of core that stores user-inserted code, and the zero table cells are used to reference these insertions (see Appendix C).

DEBUG CONTROL

Control can be given to Debug in the following ways:

1. A direct call to Debug.
2. The execution of a snapshot.
3. An unsolicited key-in of DE.
4. The Debug execution card (!XED).
5. Control Panel Interrupt (see Foreground Capability, above).

A direct call on Debug is a user-coded request for Debug to read a command. The call has the form

```
RCPYI   P,A
B       D:KEY or D:CARD
```

When the entry is D:KEY, Debug prints the message

```
!!DKEYIN
```

A Debug command will then be read from the proper device-file number assigned at SYSGEN.

Note that after the initial direct call on Debug a foreground task will have to exit in order to move Debug to a higher interrupt.

D:KEY, D:CARD, D:SNAP (snapshot) are small reentrant routines that actually trigger Debug. An unsolicited key-in during Debug will not harm the user's environment. The !XED command performs the same function as the !XEQ command except that Debug is called via D:KEY before executing the user's program.

DEBUG COMMANDS

After Debug has control, it interprets the following commands:

Code	Function
D	Define
I	Logically insert code
S	Insert snapshot
X	Step (move) snapshot
R	Remove snapshot or insertion
T	Perform selective dump on keyboard/ printer and Debug output device
P	Perform selective dump on Debug output device
C	Set Debug input device to the card reader
K	Set Debug input device to the keyboard/ printer
M	Modify memory
B	Branch (i.e., return control to program)
E	Exit from interrupt level
Q	Terminate Debug
G	Global symbol table pointer

Debug uses M:READ and M:WRITE for input/output; and hence the keyboard character NEW LINE terminates a line, EOM deletes a line, and cent (¢) deletes the previous character. Debug interprets the semicolon character (;) (if not in the message field of a snapshot) as a continuation character. The semicolon will terminate the line (or card and continue the command to the next line (or card). Blanks are ignored except within the message field of a snapshot.

Most Debug commands specify registers and memory locations. Registers are specified as follows:

RP	Program address register
RL	Link address register
RT	Temporary register
RB	Base address register
RX	Index register
RE	Extended accumulator
RA	Accumulator
RR	All of the above

Locations are specified in one of the following forms:

1. One to four hexadecimal digits.
2. SNAME, where NAME is an IDNT and its value is the load origin of such module. The Overlay Loader D option must be invoked if the user is to use IDNT names with Debug.
3. Sums or differences of values of either of the above two forms.

Examples:

```
A14
SSQRT
ABC+SSUB1+1492
SSUB1 - SSUB2
```

If the SNAME option is invoked, the user must define an insertion block (see the Debug Define command, below), and the last K:BLOCK words of the insertion block are used as a buffer for the IDNT names.

D (Define)

The Define command is used to define an insertion block when the Debug commands S or I or the SNAME option is to be used.

The form of the Define command is

```
D [start,end][,CP]
```

where

start is the memory location of the first cell of the insertion block.

end is the memory location of the last cell of the insertion block.

CP is an optional request to move Debug to the Control Panel Interrupt level. The default level is the RBM Control Task level. An unsolicited key-in of FG must be in effect when the level is specified.

I (Insert)

The Insert command designates the insertion of one or more instructions logically before (IB), after (IA), or replacing (IR) the instruction at the designated location (loc).

The form of the Insert command is

I{B,A,R} loc,inst$_1$,...,inst$_n$

where

IB designates Insert Before

IA designates Insert After

IR designates Insert Replace

The instructions may be designated in one of the following forms:

1. op*loc

   where op is a two-digit hexadecimal value representing the operation code and address modification. The second digit (i.e., address modification) must be one of the following:

   0 designating direct addressing

   2 designating indexing

   4 designating indirect addressing

   6 designating indirect addressing and indexing

   This instruction form relieves the user of creating the actual address structure for Sigma 2/3. It does not apply to the conditional branch instruction (operation code 6) nor to the register copy instructions (operation code 7). Debug will actually expand an instruction designated in this form into more than one instruction; for example, 82*1492 will expand into

```
8E02    LDA     *$+2,1
4802    B       $+2
1492    DATA    X'1492'
```

   See "Debug Expansion of Instructions", later in this chapter, for a description of the expansions.

2. 6x*loc

   where x designates the desired conditional branch; for example, 6E*1492 designates a BAN 1492 and will expand into

```
6E02    BAN     $+2
4803    B       $+3
4C01    B       *$+1
1492    DATA    X'1492'
```

   See "Debug Expansion of Instructions", later in this chapter, for a description of the expansions.

3. hex value

   which is inserted with no expansion.

4. Any mnemonic copy instruction in the Sigma 2 and Sigma 3 Computer Reference Manuals. The comma between the register specifications must be omitted.

The results of an insertion are defined in "Debug Expansion of Instructions", later in this chapter.

An example of the insert command is as follows:

```
IB $SUB+1000, 80*$SUB+25, 75A1, 40*$SQRT+0,;
RCPYIPL,ROR*LT,REOR XB
```

S (Insert Snapshot)

The Insert Snapshot command inserts (in the same manner as the instruction Insert Before) a snapshot at the designated location so that when control passes through loc, the following transpires prior to executing the instruction that was at loc:

1. The optional conditions are evaluated, and if false, the snapshot is bypassed.

2. If the conditions are true (or if none are specified), the following is output:

   SNAP AT loc

   message (if any)

   followed by the designated dumps.

Such output is always transmitted to the Debug output device; and if any of the dumps designate the keyboard/printer, then the SNAP and the message line also will be transmitted to the keyboard/printer. A user can make a maximum of 32 snapshot and instruction insertions (see "Debug Insertion Structure", later in this chapter, for the calling sequence for a Snapshot command.) The form of the Insert Snapshot command is

S[K,S] loc[/conditions/]['message'][,dump requests]

where

S is a request to snapshot and resume execution.

SK is a request to snapshot and transfer control to the keyboard/printer for Debug input.

SS is the same as SK, but may be stepped (see Debug command X).

conditions, message, dump requests are as described below.

Conditions. The format of the conditions is

$$r_1 \left\{\begin{matrix}\& \\ |\end{matrix}\right\} r_2 \left\{\begin{matrix}\& \\ |\end{matrix}\right\} r_3 \cdots \left\{\begin{matrix}\& \\ |\end{matrix}\right\} r_n$$

where r_i is a relational expression of the form

$$\left\{\begin{matrix}\text{loc} \\ \text{constant} \\ \text{register}\end{matrix}\right\} \quad [*] \quad \begin{matrix}= \\ < \\ > \\ <= \\ >= \\ <>\end{matrix} \quad \left\{\begin{matrix}\text{loc} \\ \text{constant} \\ \text{register}\end{matrix}\right\}$$

where constant is the same form as a loc preceded by a #; for example,

#1492 or #SSUB+57

The meaning of the operations in hierarchical order are as follows:

=	equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal
&	logical and
|	logical or

The comparison is arithmetic unless the operator is preceded by an asterisk (*), in which case the comparison is logical.

Message. Message is a string of any EBCDIC characters except quote (').

Dump Requests. The format of the dump requests (if any) is

$$[T] \left\{\begin{matrix}\text{register} \\ \text{loc} \\ \text{loc ... loc}\end{matrix}\right\},,, \quad [T] \left\{\begin{matrix}\text{register} \\ \text{loc} \\ \text{loc ... loc}\end{matrix}\right\}$$

where T designates a particular dump to be output on both the keyboard/printer and the Debug output device. If T is absent, the dump will be output to the Debug output device only. Only one dot (.) is necessary in specifying a block of memory locations. Extra dots are ignored.

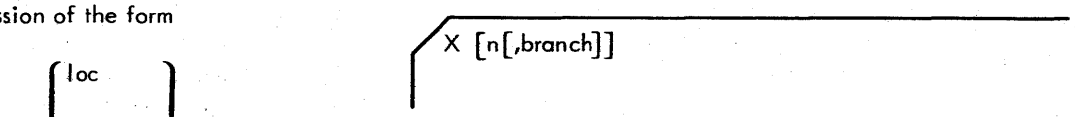
An example of the snapshot command is as follows:

```
S$SUB+505/RA=#   0&1492<1496/'TAB1   FULL',
STAB1...$TAB1+256,   RR
```

X (Step Snapshot)

If control is at the Debug input device as a result of a stepping snapshot (SS), the X command moves the snapshot to memory location n, keeping the same conditions, message, and dump requests. Control is then transferred to the branch location.

The form of the Step Snapshot command is



where

n is the memory location.

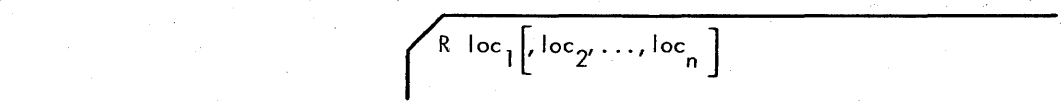
branch is the branch location.

If the snapshot was executed at location ALPHA, the default cases are branch = ALPHA and n = ALPHA+1.

R (Remove Snapshot or Insertion)

The Remove command restores the displaced instruction to its original memory location. The command releases the zero table entry and, if the entry is the latest snap or insertion, releases its space in the insertion block. Note that the space in the insertion block is regained only if the Remove command affected the latest entry in the insertion block.

The form of the Remove command is

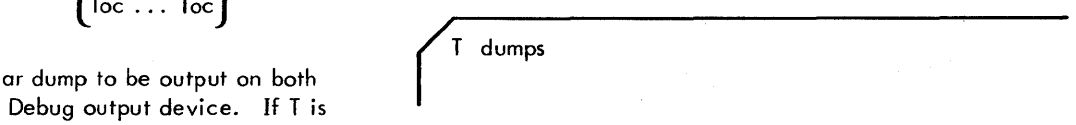


where loc is the memory location.

T (Selective Dump on the Keyboard/Printer and the Debug Output Device)

The T command outputs the contents of the requested locations and registers in hexadecimal on both the keyboard/printer and the Debug output device. Console interrupt will transfer control to the keyboard/printer after the current line is output.

The form of the T command is



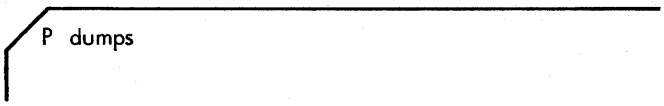
where dumps (i.e., dump requests) have the following forms (there can be several dump requests in any order separated by commas):

loc	SSUB+3
loc ... loc	$SUB ... 3FFF
register	RA
all registers	RR

P (Selective Dumps on the Debug Output Device)

This command is identical to the T command except that the dumps go only to the Debug output device.

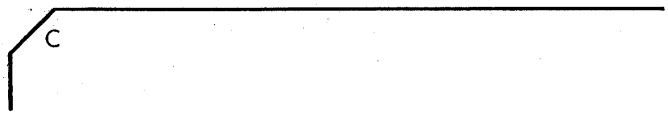
The form of the P command is



C (Debug Input Device)

The C command gives control to the Debug input device.

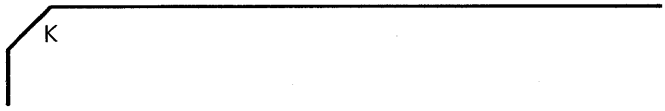
The form of the C command is



K (Keyboard/Printer)

The K command gives control to the keyboard/printer.

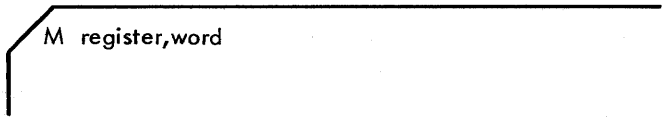
The form of the K command is

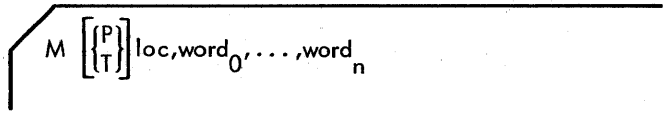


M (Modify Memory)

The M command modifies memory locations or registers.

The form of this command may be either of the following:





where

loc is the first memory location to modify.

word$_i$ is the hexadecimal value (or mnemonic register operation; see item 4 under the Debug I command) to be stored in the designated register or at location loc+i.

P if present, is a request to print the hexadecimal value of the effective location, its previous value, and its new value.

T if present, is a request to type the hexadecimal value of the effective location, its previous value, and its new value.

Examples of the M command are

1. M$SUB+1, 4, 1, $SUB+2, RADDIZE

   where the following cells are modified if SUB is located at 100_{16}:

Loc	Value
0101	0004
0102	0001
0103	0102
0104	7C68

2. MRA, $SUB

   This sets the A register to 0100. Note that an MRP command will change the program address portion of the program status doubleword.

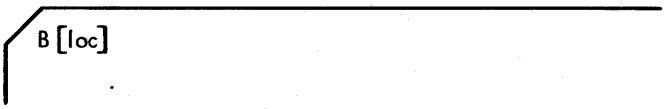
3. MT 149A, RCPYIPA

   This will produce the following output if the contents of location 149A was FFFF prior to the command 149A: FFFF → 75F1.

B (Branch)

The Branch command allows the user to insert loc into the program address portion of the program status doubleword and to exit from Debug. If loc is not present, the user just exits from Debug.

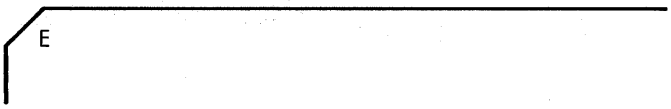
The form of the Branch command is



E (Exit From Interrupt Level)

The E command allows the user to force an unusual exit from the highest active interrupt level below Debug. Debug will still have control after this command.

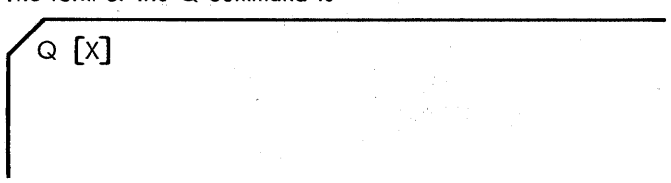
The form of the E command is



Q (Quit Debug)

The Q command causes Debug to reset its internal flags and zero table cells, restore RBM's original interrupt level, trigger the Job Control Processor, and exit. If the X option is present, Debug will also disconnect (i.e., unload) itself from the system.

The form of the Q command is



G (Global Symbol Table Pointer)

The G command specifies the first location of a symbol table separately created at assembly time, or by the use of modify (M) commands. The symbols may be used in any of the commands in place of a location or a value by preceding the symbol with an @ sign. The symbol is assigned its corresponding value as part of command processing. The symbol table is composed of a set of five-word entries for each symbol, followed by one word set equal to zero. Each symbol must be left-justified and padded with blanks for a total of eight characters. The value of the symbol is placed in the fifth word.

The form of the G command is

```
G start
```

where start is the location of a symbol table.

DEBUG ERROR MESSAGES

Error messages are shown below:

Message	Meaning
ERROR SYNTAX	Syntax error
ERROR COMMAND	Command error
ERROR FOREGRND	Command attempts to affect foreground without a hardware interrupt level specified for Debug (see Debug D command)
ERROR OVERFLOW	Either insertion block or zero table overflow
ERROR IN/OUT	Input/output error

When Debug encounters an error, it aborts a background job if there is no !ATTEND card. Otherwise it requests further commands from the keyboard/printer. At this time, Debug will not have modified the environment, allowing the user to attempt recovery. (It is assumed that the user will respecify any erroneous commands.)

A KEYIN error message issued as the result of an unsolicited key-in of DE, or an abort code of DE issued as the result of a direct call on Debug, implies that Debug is not part of the system. This can be corrected by queueing in Debug (i.e., an unsolicited key-in of Q DEBUG).

DEBUG EXPANSION OF INSTRUCTIONS

EXPANSION OF INSERTED INSTRUCTIONS

Class 1 instructions that are inserted via the insert (I) command are expanded into more than one instruction if designated in the op *address form. (Note that expansions of indirect instructions are not reentrant.)

Op is direct (0):

```
op      *$ + 2
B       $ + 2
DATA    address
```

Op is indexed (2):

```
op      *$ + 2, 1
B       $ + 2
DATA    address
```

Op is indirect (4):

```
STA     $ + 6
LDA     *$ + 7
STA     $ + 5
LDA     $ + 3
op      *$ + 3
B       $ + 4
DATA    0
DATA    0
DATA    address
```

Op is indirect and indexed (6):

```
STA     $ + 6
LDA     *$ + 7
STA     $ + 5
LDA     $ + 3
op      *$ + 3, 1
B       $ + 4
DATA    0
DATA    0
DATA    address
```

Class 2 instructions are expanded as follows:

```
op      $ + 2
B       $ + 3
B       *$ + 1
DATA    address
```

EXPANSION OF MOVED INSTRUCTIONS

An instruction that is moved from the point of insertion to the insert block will require expansion if its addressing is relative or if it is a register copy instruction in which the P register is the source.

The relative instructions are expanded the same as the inserted instructions discussed in the first part of this appendix. In the case of Insert Before (IB) or snapshots, register copy instructions in which P is the source and the clear bit is set will be expanded in one of two ways:

1. If the destination is the A register:

```
LDA     $ + 3
op      A, A
B       $ + 2
DATA    α + 1
```

2. If the destination is not the A register:

```
STA     $ + 5
LDA     $ + 5
op      A, R
LDA     $ + 2
B       $ + 3
DATA    0
DATA    α + 1
```

In the above expansions, α is the location (point) of the insertion and op has the appropriate settings for the incrementation and inversion bits.

Debug has no facility for expanding a copy instruction where either (1) the P register is the source, the A register is the destination, and the clear bit is reset, or (2) the P register is the destination and the clear bit is reset. In this case a Debug syntax error is generated.

DEBUG INSERTION STRUCTURE

An insertion at location α will result in the following:

```
α     B       *β
β     DATA    γ
```

γ {
- moved instruction expansion if IA command
- inserted instructions or snapshot call code
- moved instruction expansion if IB or snapshot command

```
      B       *$ + 1
      DATA    α + 1
```

where β is one of the Debug locations in the zero table and γ is an area in the insertion block.

DEBUG SNAPSHOT CALLING SEQUENCE

A snapshot inserted at location α will generate the following calling sequence (which is inserted in the insertion block similar to a Debug IB command):

```
α1      DATA                D:SNAP
α2      DATA                block
        instruction that was at location α
entry   WD                  X'FC' (foreground only)
        STA                 *α2
        RCPYI               P, A
        B                   *α1
        DATA                α
        DATA                key
        conditions if any
        DATA                -1
        message if any
        DATA                -1
        dumps if any
        DATA                -1
        expanded instruction from location α
        B                   *$ + 1
        DATA                α+ 1
```

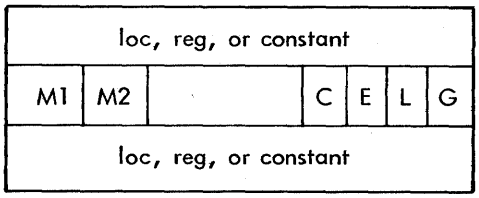
where

block is the address of the first word of the insertion block and is used to save the A register.

key (bits 0-2) designates type of snapshot: setting bit 0 designates stepping snapshot; setting bit 1 designates line printer snapshot output; and setting bit 2 designates keyboard control requested.

message is the string of EBCDIC characters, if any.

condition is a string of relational expressions separated by logical operators. A relational expression occupies three words as follows:



where

M1 (bits 0-1) designates the type of quantity in the first word:

- 00 location
- 01 register
- 10 constant

M2 (bits 2-3) designates the type of quantity in the third word.

C (bit 12) designates comparison where 0 = arithmetic and 1 = logical.

E (bit 13) designates equal comparison.

L (bit 14) designates less than comparison.

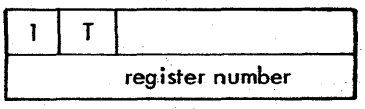
G (bit 15) designates greater than comparison.

A logical operator occupies one word:

0 logical or

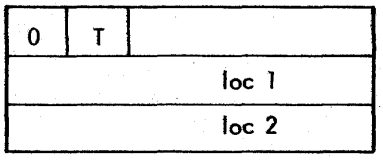
1 logical and

dumps are two-word or three-word items:



register dump

or



memory dump

where

T = 1 designates keyboard/printer and line printer output.

T = 0 designates line printer output.

A zero register number designates all registers.

13. BASIC SPOOLING SYSTEM

PURPOSE

The Basic Spooling System (BSS) provides the following:

1. Allows programs to execute at a speed not limited by peripheral speeds by providing a disk buffering file during periods of high peripheral utilization.

2. Through use of the disk buffer file, BSS will maintain efficient peripheral utilization by smoothing the peaks and valleys of peripheral usage thereby driving peripheral devices at or near rated speed.

3. Resolves contentions for a peripheral device between foreground and background or between foreground tasks themselves such as XSP and IDEN by spooling output from one or all of the conflicting tasks.

4. A convenient point-to-point foreground utility utilizes a disk buffer file to synchronize speed and availability of peripheral devices.

IMPLEMENTATION PHILOSOPHY

A capability is provided whereby tasks may output through conventional operational labels or FORTRAN device unit numbers and merely through reassignment (or default assignment), have the output directed to an intermediate spooling file. No modification to foreground or background tasks is required.

The disk buffering employed utilizes conventional RBM random files and standard RBM I/O to provide a low overhead, high reliability spooling system. The disk allocation is circular in nature with output occurring in a first in, first out (FIFO) fashion.

BSS itself operates as a resident, semiresident or nonresident foreground task. Multiple copies of BSS may be used to provide multiple concurrent spooled operations. A simplified overview of BSS operating as a line printer spooler is shown in Figure 10.

BSS is implemented as a foreground task which reads from a foreground operational label and writes to a circular disk spooling file. Concurrently BSS reads from the disk file and outputs through an operational label to a physical device. It is through the use of RBM-16 logical devices (Version G00 and later) that the user task's output operational label is connected to the BSS input operational label. A detailed overview of a BSS line printer spooler is shown in Figure 11.

Since the flow of data is initiated and terminated by conventional RBM operational labels, many variations of BSS are possible as illustrated in Figure 12.

LOADING BSS

The control cards for allocating the file and loading BSS are:

```
!JOB
!PAUSE      KEY-IN     'SY,S'
!RADEDIT
!#ADD     UP,COPY,ALL,,R,SY
!#END
!ASSIGN  OV = COPY,UP
!OLOAD     ,F
!$TCB     +C30C,+1200
!$ROOT     ,,BI,3
!$END
!RADEDIT
!#TRUNCATE     UP,COPY
!#END
!FIN
```

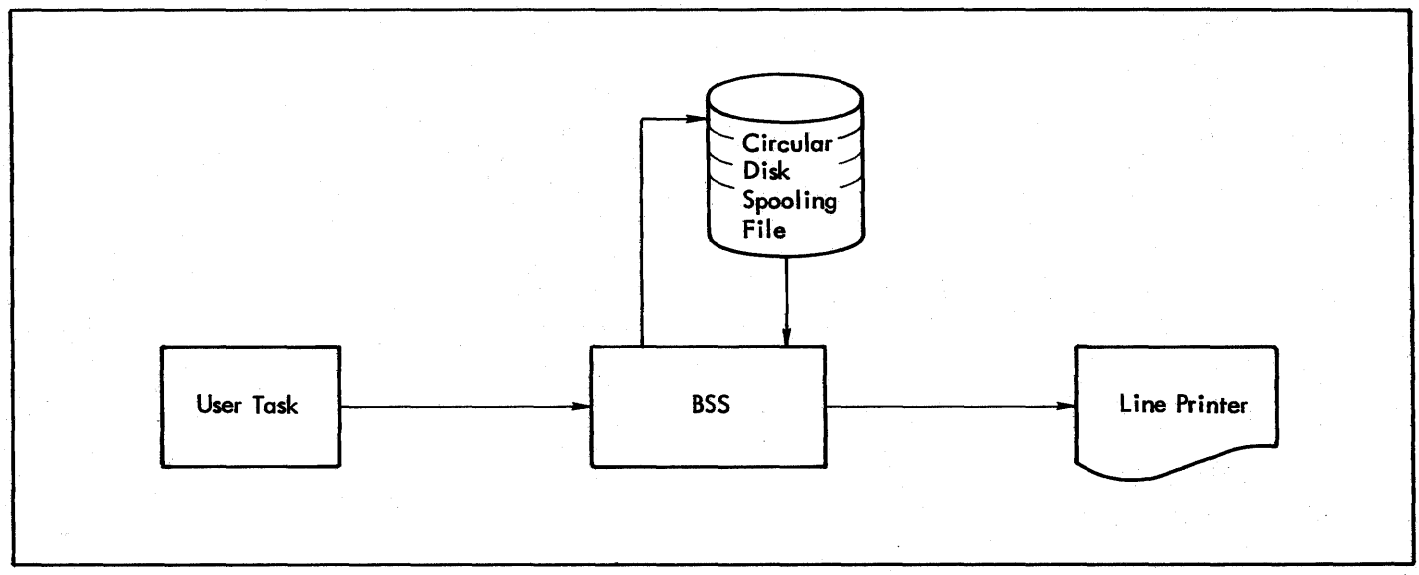


Figure 10. Simplified Overview of Line Printer Spooler

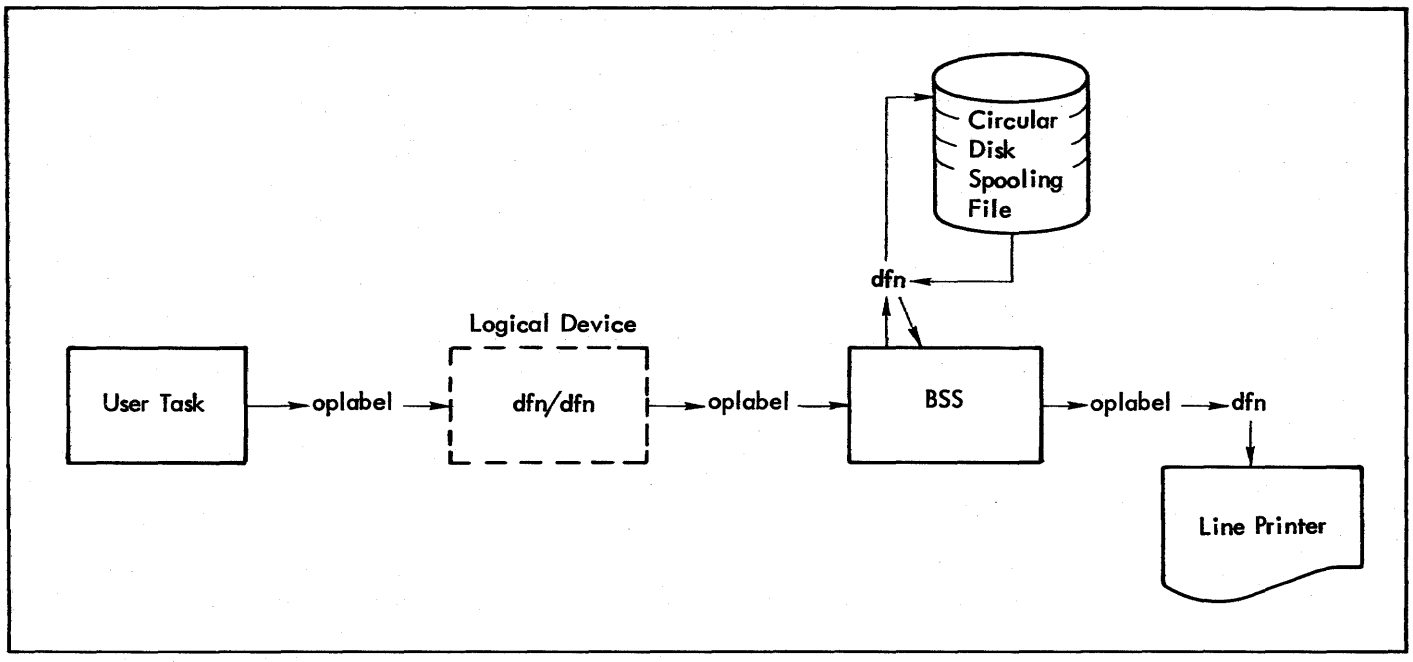


Figure 11. Detailed Overview of Line Printer Spooler

With this job stack, BSS will be loaded into a permanent file named 'COPY', in the 'UP' area. BSS, when invoked, as with a Q COPY keyin, will run at interrupt level X'1OC' (interrupt level is user specified).

Note: The 'Q' keyin can only be used to load programs in the 'UP' area.

ALLOCATING SPOOLING FILES

A spool file must be allocated in the 'SD' area. Care must be taken in naming the spool file. For example, if BSS will spool data to the CP foreground oplabel and the default spooling file name is being used, the first two characters of the file name must be 'CP'. The remaining six characters must be 'SPOOL'. The control cards for allocating a CP spool file and initiating the copy are:

```
!JOB
!RADEDIT
!#ADD SD, CPSPOOL, 100, , R
!#END
```

Note: The spooling file must be in the 'R' format. Consult Table 21 for spooling file size requirements.

INITIATING BSS

BSS may be brought into core as a resident foreground task at boot time, or as a semiresident or nonresident foreground task through use of the background job stack, or through use of the 'Q' keyin (assuming BSS resides in the 'UP' area).

BSS requires certain fundamental information in order to initiate an operation, namely the source of the data, the destination for the data, and the name of the intermediate spooling file to be used. The relationship of the destination operational label and spooling file name is 'op SPOOL', 'SD' where op is the destination operational label. The default association can be overriden by supplying any spooling file name through assembly or load time options.

If BSS has been assembled with the source, destination, and spooling file defined, then no operator intervention is required to initiate operation once BSS is loaded.

Lacking sufficient information, BSS will query the operator for source and destination operational labels. If the source has not been defined by assembly BSS will prompt with:

```
# SPECIFY   INPUT
```

to which the operator may respond with a two-character foreground operational label to be used for BSS input.

If the destination has not been defined by assembly or if BSS will prompt with:

```
# SPECIFY   OUTPUT
```

to which the operator may respond with a two-character foreground operational label. BSS will inform the operator

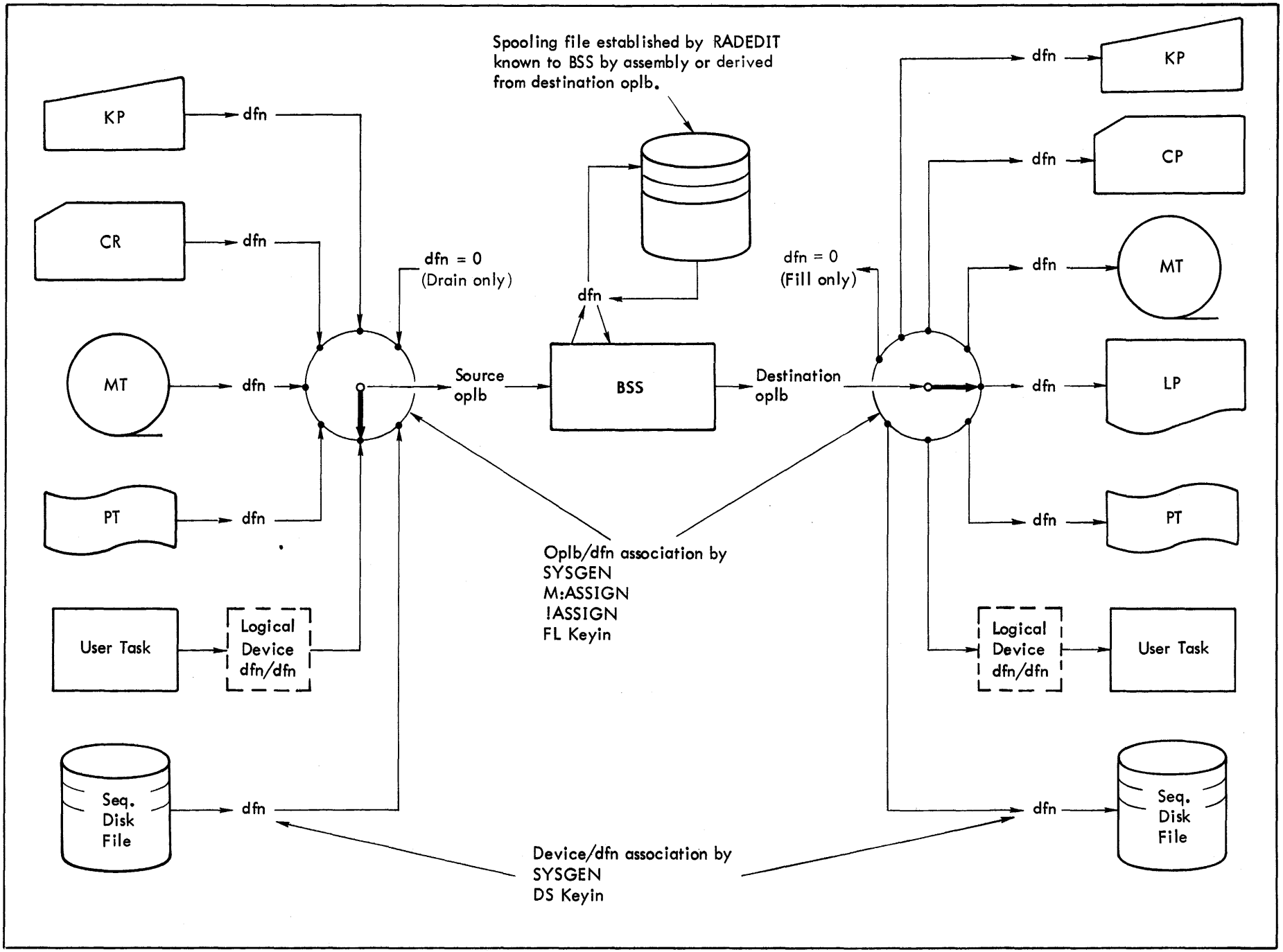


Figure 12. Variations of Basic Spooling Systems

Table 21. Spooling Volume Requirements

Device Type	Model Number	Rated Speed (Rec/Min)	Comp. Char. Per 10 Minutes Operation	Sectors Required Per 10 Minutes Operation			
				Model 7204	Model 7242	Model 7250	Model 32xx
Readers	7121	200	80K	225	79	225	310
	7122	400	160K	450	160	450	625
	7140	1500	600K	1700	580	1700	2350
Printers	7450	225	106K	295	100	295	415
	3451	350	175K	490	170	490	680
	7440	800	400K	1100	390	1100	1550
	7445	1000	500K	1400	490	1400	1950
Punches	7165	100	40K	110	390	110	155
	7160	200	80K	225	79	225	310

Assumptions:
1. 50% overall data compression.
2. 80-byte records.
3. 100-byte print records.

of its last word address, so that the first available address for the loading of subsequent foreground programs may be determined. However, the last word address will not appear if Data Switch 2 is set.

BSS OPERATION AND CONTROL

The copy process will proceed immediately after BSS is initiated and the source, destination, and spooling file are known.

FORMS CONTROL

When a *FORM record is encountered in an output stream, control will be transferred to the Forms Control Module. Upon entry, the 'A' register will contain the retum address to BSS and the 'X' register will point to the following argument list

0 X'3005'

1 'OC'

2 Address of 'FORM' record

3 Byte length of 'FORM' record

Upon exit the X-register has the following significance:

X = -1 STOP the stream

X = -2 SKIP the stream

Other CONTINUE the stream

The delivered Forms Control Module will output the *FORM record to 'OC' and exit with X = -1 to STOP the stream.

The purpose of isolating this code is to allow installation to conveniently add installation dependent code (e.g., special forms routing).

BSS can be controlled with the following keyins:

#GO	xx	Start operation to foreground oplabel xx. Normally, this keyin will only be required after a #STOP or #LOCK keyin.
#STOP	xx	Suspend operation to foreground oplabel xx. Handy for verifying output.
#LOCK	xx	Same as #STOP, but takes effect at the conclusion of the current file being output. Furthermore, if a #STOP is in effect, this keyin is an implied #START. LOCK only applies to the output oplabel.

#SKIP	xx	Skip forward to next EOF in output stream. Data skipped in this fashion will remain in the spool file.
#BACK	xx	Restart output for operational label xx at the backup point. The backup point is the spooling file holdback point (defaults to five granules or previous EOF, whichever is greater).
#TERM	xx	Terminate the BSS task associated with output operational label xx. BSS will automatically terminate a copy operation when two consecutive EOFs are read. All operations are ceased, the spooling file directory is updated and an M:TERM is performed.
#BIN	xx	Perform "write binary" to output oplabel.
#EBC	xx	Perform "write EBCDIC" to output oplabel.
#IVFC	xx	Append single space vertical format byte to output oplabel.
#FIN	XX	Flush out any Data Remaining in Spool File

ABORT CODES

If the spool file cannot be found, BSS will abort with code #F.

If any fatal errors occur, BSS will abort with the following codes:

#I	Error on input
#O	Error on output
#F	Error on spooling file

BSS will not abort if operator intervention is required on the output or input oplabel. Instead, BSS will output

##STOPPED xx

and simulate a #STOP keyin. The operator may correct the problem and then keyin #START.

ASSEMBLY OPTIONS

Various options can be included in BSS at assembly time, once the source deck is extracted from the standard release tape. These options are described in the Technical Manual (and the source listing) and control such items as default input and output oplabels as well as spool filename and area.

The cost for each concurrent spooled operation is as follows:

1K	core for BSS + 2* spooling file block size + 2* record size (defaults to 140 bytes).
3	foreground DFNs for spooling file access, input and output.
2	foreground operational labels.
1	spooling file (R format). See Table 21 for file size requirement.
1	foreground interrupt level lower in priority than IO.
1	foreground or background DFN for user task.
1	foreground or background operational label for user task.

APPENDIX A. ADDITIONAL RBM PROCESSORS

A set of subsystems and processors is distributed with RBM on the transmittal tape. The Overlay Loader, RAD Editor, Utility, and Debug processors are described in Chapters 7, 8, 9, and 12 respectively. XSYMBOL, FORTRAN, ANS FORTRAN, and the FORTRAN library are described in their own individual manuals (see the Related Publications page of this manual).

The following additional processors are available on the transmittal tape and are described in this appendix:

Name	Purpose
PLOT	A symbiont subsystem for the 7530 or 7531 graph plotter. (Catalog No. 705780.)
INDUMP	A stand-alone DUMP program to be used in conjunction with RBM.
COMPRESS	A processor for creating blocked compressed EBCDIC files on tape, used in preparing the source and listing files on the transmittal tape.
EXPAND	A processor for expanding the blocked, compressed files created by COMPRESS to files composed of either 80-byte source records or 134-byte line printer listing records.
REPLACE	A processor for replacing monitor overlays, useful in system maintenance (described in the RBM/SM Reference Manual, 90 30 36).

SYMBIONT PLOTTING SYSTEM

The symbiont plotting system performs circular buffering of plotter commands in a RAD or disk file (PLSYMB). A set of background subroutines in the FORTRAN subroutine library is provided to build the file. The background subroutines trigger a foreground task that reads the file and drives the plotter. The trigger is accomplished via a public library subroutine. A set of unsolicited operator key-ins permit the operator to supervise the plotting operation.

UNSOLICITED OPERATOR KEY-INS FOR PLOT

Key-in	Effect
PLPR (INIT)	Report the amount of RAD space left in the plot file. (Until the end-of-file is encountered, the amount of space used will be reported.) After the plot data have wrapped around, the amount of unused space will then be reported.
PLHA (LT)	Stop plotting immediately.
PLAB (ORT)	Stop plotting immediately and discard plot data to the beginning of the next plot and halt.
PLST (ART)	Stop plotting.
PLSU (SPEND)	Stop plotting at the end of the current plot.

These key-ins have no effect on the background job.

BASIC PLOTTER CONTROL SUBROUTINES

This group of four subroutines in the FORTRAN library gives the assembly language programmer all of the functions necessary to draw a plot. They are used by other programs that give the programmer more sophisticated functions to simplify the task of making a plot. The subroutines generate plotter control data and transmit it to the RAD by a call to the Monitor I/O. A foreground program is then started that reads the data from the RAD and writes it on the plotter.

If a call is executed that would move the pen off the paper, the call is ignored. It is assumed that the pen starts one-half inch from the minus-Y edge of the paper (the right border of the paper roll.)

ENTRY POINTS

RCPYI P,L

B PENUP (register T is changed)

This entry will cause the pen to be raised if it is down.

RCPYI P,L

B PENDN (register T is changed)

This entry will cause the pen to be lowered if it is up.

RCPYI P,L

B INITIAL (only register B is saved)

The current pen position is set to X=0, Y=0, and this position will now be the new plotter reference point. Accumulated data is output at this time. Note that at any time there may be a partial buffer of plot data that has not been transmitted to the device. Therefore, "INITIAL" must be entered at the end of the plot job to ensure the completion of the plot.

RCPYI P,L

B MMVE (register T is changed)

X is in register E and Y is in register A. The pen is moved along an approximation to a straight line from its current position to the new location X,Y. X and Y are fixed

points, the number of increments from the reference point that is normally the lower left corner of the plot.

See the FORTRAN library description for a description of FORTRAN calls to higher-level PLOT subroutines.

INDUMP

INDUMP is a "stand alone" dump facility that provides a printed record of the contents of memory when the RBM postmortem dump, operator key-in dumps, or the DEBUG dumps cannot be used.

INDUMP LOADING TECHNIQUES

RESIDENT FOREGROUND

INDUMP may be loaded into the resident foreground area by the usual techniques. It requires 600_{16} memory locations. The last 200_{16} memory locations may be overwritten if the command to display the file control tables in expanded form is not to be used.

RESIDENT HIGH MEMORY

If the memory size in SYSGEN is indicated to be other than a multiple of 8192, INDUMP will automatically be moved to K:UNAVBG (beginning of "unavailable" background memory) when loaded into the foreground. The space initially occupied by INDUMP in the foreground area may then be overwritten.

SELF-LOADING

A version of INDUMP may be prepared that can be loaded using the hardware bootstrap from cards, magnetic tape, or paper tape. To do this, the REL version of INDUMP is loaded and executed with a !XEQ card with the following parameters, which will generate the self-booting version on the BO device:

```
!XEQ la,ma,fa,ba,cc,oc,lo,nl,ls
```

where

- la is the load address.
- ma is the RBM last word address.
- fa is the foreground last word address.
- ba is the background last word address.
- cc is the channel and device number of the boot device (format: ccdd).
- oc is the channel and device number of the keyboard printer (format: ccdd).
- lo is the channel and device number of the line printer (format: ccdd).
- nl is the number of lines per page (37 or 52).
- ls is FFFF for a low-speed line printer and 0 for a high-speed line printer.

All parameters are four-character hexadecimal quantities except nl, which is decimal.

If only la is given, the other parameters will be picked up from the RBM system, which punches the self-booting version.

INDUMP OPERATIONS

INDUMP may be used to provide snapshots of the registers and core when DEBUG cannot be used. The call has the form

```
        RCPYI   P,L
        B       INDUMPFWA + 1   { FWA = first word address
        DATA    LOWLIM
        DATA    HIGHLIM
RETURN
```

INDUMP may be called to permit the operator to type in commands to it using the calling sequence

```
        RCPYI   P,L
        B       INDUMPFWA
RETURN                          RETURN ON GO command
```

INDUMP may be started from the console in the event of system failure. The procedure is as follows:

1. Move the COMPUTE switch to the IDLE position.
2. Copy the values of the P register and PSW (these will be input later using the PI command).
3. Place the start address of INDUMP in the data switches.
4. Select the S register with the register SELECT switch.
5. Move the CLEAR/ENTER switch to CLEAR; then enter.
6. Place the STORE/FETCH switch in the FETCH position and the ADDRESS HOLD switch in the HOLD position.
7. Momentarily move the COMPUTE switch to the STEP position (3 only).
8. Move the STORE/FETCH and ADDRESS HOLD switches back to NORMAL.
9. Move the COMPUTE switch to RUN.

After INDUMP is started, it will type out the message ENTER LIMITS. The operator can then respond with a command of the form

command [hex-value, hex-value]NL

where command may be

DM Dump RBM area.

DF Dump Foreground area, including Public Library.

DB Dump Background area.

DA Dump all of core up to K:UNAVBG.

ZM Zero RBM area, including Public Library.

ZF Zero Foreground area.

ZB Zero Background area.

ZA Zero all of core up to K:UNAVBG.

PI Place the first hex value in the stored P register location and the second in the stored PSW location.

GO Restart operation of RBM with values given by PI command or obtained from call.

DT Dump file control tables.

After a dump or zeroing between limits, the ENTER LIMITS message will be retyped and a new command may be entered. If hex values are specified on any command, they will override the command's implicit limits.

COMPRESS

The COMPRESS processor reduces the length of the RBM distribution tape. COMPRESS reads records from the SI file or device, expands them to 134 bytes by filling with blanks on the right, then blank-compresses and blocks those records into 1024-byte blocks. It then outputs those blocks to the CO device file.

EXPAND

EXPAND takes the blocked records from CI formed by COMPRESS and generates either 80-byte source records or 110-byte listing record on the EO device file. The EXPAND processor is invoked by

```
!EXPAND {S/L}, ident[, n]
```

where

S indicates that the source records are to be written on the EO operational label.

L indicates that the listing records are to be written on the EO operational label.

ident is a 1-8 character identifier that exists beginning in column 73 of the source file.

n indicates the number of files to be expanded.

The operational labels CI (Compressed Input) and EO (External Output) must be assigned to the appropriate device file numbers before executing EXPAND.

The EXPAND processor searches the CI device until it finds the specified ident. If a double EOF is encountered, the tape will be rewound and a second search made. If the ident is not found on the second search, an appropriate message is output on OC.

When the selected ident is located, the file is decompressed and output on the EO device according to the selection parameter S or L.

For example, the command

```
!EXPAND   L,TOC
```

will cause the EXPAND processor to list the TOC (Table of Contents file on EO. Similarly, the command

```
!!EXPAND   L,EXP,3
```

will cause the EXPAND processor to list three files, beginning with EXPAND.

To obtain a source magnetic tape, assign EO to a magnetic tape and input

```
!EXPAND   S,EXP
```

which will produce a source file acceptable as symbolic input to XSYMBOL.

The following table should be used to determine the standard assignments for an installation's background operational labels and to determine which operational labels, if any, should be suppressed by being assigned to file 0. The standard operational labels are defined under the !ASSIGN command in Chapter 2.

Processor \ Operational Label	Device Number 1	CC	SI	UI	AI	BI	BO	UO	LL	DO
RBM (Job Control Processor)	Read/Write unsolicited key-in	Read Control Commands			Read Absolute Binary	Read Object modules with !REL command			Write Control Command Images	
XSYMBOL[†]			Read Source Statements	Read Update Records			Write Reloc. Binary		Used for CC Diagnostics	Write XSYMBOL Error Messages[††]
Concordance			Read Source Statements							Write Concordance Error Messages[†††]
Basic FORTRAN IV or ANS FORTRAN IV			Read Source Statements				Write Reloc. Binary			
Math Library										Write Library Error Messages
Overlay Loader		Read Control Commands							Log Control Commands	Write Loader Error Messages[†††]
RAD Editor		Read Control Commands				Object Module Input to System and User Libraries	Output Copies of Object Modules from System and User Libraries		Log Control Commands	Write Error Messages and operator key-ins
Utility Executive			Read Control Commands						Log Control Commands	Write Utility Error Message[†††]
Utility Copy[††††]			Read Control Commands	Read Input						
Utility RECEDIT			Read Control Commands and Modific Input	Read Input				Write Output		
Utility OMEDIT			Read Control Commands	Read Input		Read Binary Modific. Input		Write Output		
Utility DUMP			Read Control Commands	Read Input						
Utility SEQEDIT			Read Update Data	Read Input				Write Output		

[†]Uses oplabel SO to output source statements (updated, if applicable).

[††]Suppressed if assigned to same device as LO.

[†††]Suppressed if assigned to same device as OC.

[††††]May use any oplabel for output.

Processor \ Operational Label	LO	LI	PM	OC	X1	PI	OV	X2	X3	S2	GO	X4	X5
RBM (Job Control Processor)			Write Absolute Binary Monitor (SYSGEN only)	Write Processor and Monitor Abort Messages		Read RBM Overlays	Write Program Loaded by !ABS Command				Write Object Module with !REL command		
XSYMBOL	WRITE Listing Output and XSYMBOL Error Messages			Operator Communications	Intermediate Output	Read XSYMBOL Overlays		Output Encoded Text	Output Program Locals	Output Standard Procedures	Output Execution Object Language		
Concordance	Write Listing Output and Concordance Error Messages												
Basic FORTRAN IV or ANS FORTRAN IV	Write Listing Output and FORTRAN Error Messages				Intermediate Output	Read FORTRAN Overlays					Output Execution Object Language		
Math Library	Write Library Error Messages			Operator Communications									
Overlay Loader	Write Maps	Read Reloc. Binary Library File		Operator Communications	Contains Symbol Table for each segment	Read OLOAD Overlays	Write Core Images	Read MODIR File			Read Reloc. Binary		
RAD Editor	Write Maps and Dumps of Files			Operator Communications	Replace Files and Maintain Libraries	Read RAD Editor Overlays		Replace Files and Maintain Libraries	Maintain Libraries and Update Directories			Maintain Libraries	
Utility Executive	Write Utility Error Messages, Control Command Images and other Output			Operator Communications		Read Utility Overlays							Prestore Commands From SI
Utility Copy												Input for Verify	
Utility RECEDIT	Write Modification Log												
Utility OMEDIT	Write Module Log								Prestore BI				
Utility DUMP	Write Dump												
Utility SEQEDIT	Write Listing												

APPENDIX C. SYSTEM ZERO TABLE AND CONSTANTS

Table C-1. Monitor Zero Table

Address		Name	Purpose and Assignment
Dec.	Hex.		
0	0		Reserved for Monitor Use.
1	1	K:AC	Pointer to Current Floating Accumulator.
2	2	K:AC1	Pointer to Current Floating Accumulator (1).
3	3	K:AC2	Pointer to Current Floating Accumulator (2).
4	4	K:AC3	Pointer to Current Floating Accumulator (3).
5	5	K:FFLG	Pointer to Current Floating Flags.
6	6	K:BASE	Pointer to Current Task Reentrant Temp Stack.
7	7	K:TCB	Pointer to Current Task TCB.
8	8		Reserved for Monitor use.
9 : 63	9 : 3F		Standard Constants for Foreground, Monitor, and Background Use (see Table C-2 for complete list).
64 : 99	40 : 63		IOCS Pointers and Constants.
100 : 132	64 : 84		Reserved for Monitor Use.
133	85		Debug Transfer Vector D:KEY.
134	86		Debug Transfer Vector D:CARD.
135	87		Debug Transfer Vector D:SNAP.
136 : 167	88 : A7		Reserved for Debug Use.
168 : 198	A8 : C6		Real-Time Foreground User Storage (reserved for foreground communication between foreground and background or for address literals or constants).

Table C-1. Monitor Zero Table (cont.)

Address		Name	Purpose and Assignment
Dec.	Hex.		
199 . . . 225	C7 . . . E1		Monitor Service Routines Transfer Vectors (see Table 7 for list).
226 . . . 251	E2 . . . FB		Monitor Constants (see Table C-3).
252 . . . 255	FC . . . FF		Counter Interrupt Locations (optional).

Table C-2. Standard Constants

Address		Value		Address		Value	
Dec.	Hex.	Dec.	Hex.	Dec.	Hex.	Dec.	Hex.
9	9	32768	8000	20	14	16	10
10	A	16384	4000	21	15	8	8
11	B	8192	2000	22	16	4	4
12	C	4096	1000	23	17	2	2
13	D	2048	800	24	18	1	1
14	E	1024	400	25	19	0	0
15	F	512	200	26	1A	-1	FFFF
16	10	256	100	27	1B	-2	FFFE
17	11	128	80	28	1C	3	3
18	12	64	40	29	1D	-3	FFFD
19	13	32	20	30	1E	-4	FFFC

Table C-2. Standard Constants (cont.)

Address		Value		Address		Value	
Dec.	Hex.	Dec.	Hex.	Dec.	Hex.	Dec.	Hex.
31	1F	5	5	48	30	14	E
32	20	-5	FFFB	49	31	-14	FFF2
33	21	6	6	50	32	15	F
34	22	-6	FFFA	51	33	-15	FFF1
35	23	7	7	52	34	-16	FFF0
36	24	-7	FFF9	53	35	32767	7FFF
37	25	-8	FFF8	54	36	32512	7F00
38	26	9	9	55	37	33023	80FF
39	27	-9	FFF7	56	38	65280	FF00
40	28	10	A	57	39	255	00FF
41	29	-10	FFF6	58	3A	61440	F000
42	2A	11	B	59	3B	3840	0F00
43	2B	-11	FFF5	60	3C	240	00F0
44	2C	12	C	61	3D	49152	C000
45	2D	-12	FFF4	62	3E	31	1F
46	2E	13	D	63	3F	127	7F
47	2F	-13	FFF3				

Table C-3. Monitor Constants

Address		Name	Purpose
Dec.	Hex.		
226	E2		Reserved for Monitor use.
227	E3		
228	E4	K:MASTD	Pointer to Master Dictionary.
229	E5	K:PAGE	Number of Lines/Printer Page (SYSGEN Parameter).
230	E6	K:BACBUF	Background I/O Buffer Pool FWA.
231	E7	K:BACKP	Protected Background FWA (Start of TCB).
232	E8		Reserved for Monitor use.

Table C-3. Monitor Constants (cont.)

Address		Name[t]	Purpose
Dec.	Hex.		
233	E9	K:PLFWA	Public Library FWA.
234	EA	K:RFFWA	Resident Foreground FWA.
235	EB	K:NFFWA	Nonresident Foreground FWA.
236	EC	K:BACKBG	Unprotected Background FWA.
237	ED	K:UNAVBG	Unavailable Memory FWA.
238	EE	K:BLOCK	Size of Blocking Buffer in Words (180 or 512).
239	EF	K:FEF	FORTRAN Background Error Severity (1).
240	F0	K:TVECT	Pointer to Transfer Vector Table.
241	F1	K:FWA	Legal TVECT Entries to FGD-FWA.
242	F2	K:LWA	Legal TVECT Entries to FBD-LWA+1.
243	F3	F:FWA1	TVECT FWA for T Register Check.
244	F4	K:LWA1	TVECT LWA+1 for T Register Check.
245	F5	K:OLOAD	Pointer to RBM OV:LOAD Table.
246	F6	P:CST9	Reserved for RBM use.
247	F7	K:CCBUF	Address of Control Card Buffer.
248	F8	K:NRFQ	Pointer to Nonresident Foreground Queue Table.
249	F9	K:NEXT	Next Available Sector in BT Area.
250	FA	K:PROTCT	Pointer to Protection Register Table.
251	FB	K:PMDTBL	Pointer to Postmortem Dump Table.
405	195	K:CPU	CPU Type and Hardware Options.

[t]These names are as defined in the RBM Monitor and are not system definitions. Any references to these locations by these names must be defined in the user program (e.g., K:PAGE EQU X'E5').

Relationships for Monitor Constants:

1. (K:PLFWA) = LWA+1 of RBM.
2. (K:RFFWA) = LWA+1 of Public Library.
3. (K:NFFWA) = LWA+1 of Resident Foreground
4. (K:BACKP) = LWA+1 of Nonresident Foreground.
5. (K:BACKBG) = (K:BACKP) + 39.
6. (K:CCBUF) = (K:UNAVBG) - 62.

APPENDIX D. ERROR MESSAGES, WARNING MESSAGES, AND ABORT CODES

RBM MESSAGES AND ABORT CODES

JCP CONTROL COMMAND DIAGNOSTICS

The following error messages may appear on the background DO device as a result of an error condition detected by JCP. These diagnostics supplement the abort or attend-mode error codes printed by JCP.

Message	Comments/ Associated Commands
.BK OPLB/DFN TBL FULL	ASSIGN, DEFINE, default assignments for system processors
.FG OPLB/DFN TBL FULL	ASSIGN
.ILL C:CODE	C: (Connect)
.ILL C:TCB	C: (Connect)
.ILL RAD SEQUENCE	WEOF, REWIND, UNLOAD, FBACK, FSKIP, RBACK, RSKIP
.INV COMMAND	Command not recognized as a Monitor service command, system processor, or user processor.
.INV OPLB OR DFN	ASSIGN, DEFINE, WEOF, REWIND, UNLOAD, FBACK, FSKIP, RBACK, RSKIP
.INV OPTION	An invalid option has been encountered on a Monitor service command
.NO 'FG' KEY-IN	ASSIGN, XEQ, C:
.NO 'SY' KEY-IN	WEOF, ABS, REL
.OP NOT MEANINGFUL	WEOF, REWIND, UNLOAD, FBACK, FSKIP, RBACK, RSKIP
.RAD TEMP OVERFLOW	DEFINE, default assignments for system processors

RBM ABORT CODES

The codes listed in Table D-1 are the standard background-job abort codes issued by RBM for abort conditions detected by the Monitor, JCP, RAD Editor, and Utility, and also by the Basic FORTRAN IV compiler and the Extended Symbol assembler. Note that the codes for abort conditions detected by the Overlay Loader are listed separately in Table D-3.

The abort codes appear in a standard abort message of the form

```
!!BKG xx ABORT loc zzzz
```

where

xx is the abort code.

zzzz is the location at which the abort occurred.

Table D-1. RBM Abort Codes

Code	Meaning
AE	Assignment error during loading; improper I/O assignment or invalid format.
AI	Irrecoverable I/O error on device assigned to operational label AI.
BI	Irrecoverable I/O error on BI device.
BO	Irrecoverable I/O error on BO device.
CC	Error in control cards or in sequence of job stack.
CK	Irrecoverable error while checkpointing.
CS	Checksum error from absolute or relocatable binary input.
DE	Debug not resident when requested.

Table D-1. RBM Abort Codes (cont.)

Code	Meaning
ER	Operator-recognized error condition.
ES	FORTRAN library abort†.
FC	Illegal FORTRAN control card.
FS	FORTRAN abort†.
FX	A control card was encountered in the FORTRAN source deck.
GO	Irrecoverable error on output to the GO file when using a !REL command.
HX	Illegal hex parameter.
IE	Error in input deck. (Usually, a negative ORG item has been input.)
IO	Irrecoverable I/O error.
LO	Irrecoverable I/O error on LO device.
MF	Machine fault interrupt has occurred.
NA	Nonexistent address used by background program (530 systems only).
NP	No patch area has been allocated.
OP	Operator abort, from unsolicited key-in.
OV	Problem with device assigned to operational label OV. (Normally, OV is assigned to the RAD.)
PE	Parity error in background (perhaps attempting to read from unavailable memory).
PO	The patch area has overflowed.
PU	Number of argument greater than temporary storage in M:PUSH†.
PV	Protection violation.
RE	RAD Editor abort†.
RS	Irrecoverable error during restart.
SI	Irrecoverable input error in SI device.
SQ	Sequence error in absolute or relocatable binary deck.
TL	Background program time limit exceeded.
TS	Temp stack overflow.
TY	Invalid load type in ABS deck.
UT	Utility subsystem abort†.
XE	Fatal error in loading.
XS	Extended Symbol abort†.

†After the abort code is output, the processor will exit via the RBM routine M:ABORT.

Note: The processing of the job stack is discontinued following any abort. If an "ATTEND" control command was in effect, the Monitor will enter an "idle" state. This will allow the operator to correct the problem and restart the job. If not in "attend", the Job Control Processor will read commands until a !JOB or !FIN command is encountered. All control commands encountered prior to the !JOB or !FIN command will be logged with an indication (">" will precede the command) that they have been ignored.

OVERLAY LOADER MESSAGES AND ABORT CODES

I/O ERROR MESSAGE

The I/O error message has the following format. It is followed by a "!!BKGD IO ABORT..." message

** oplb device type and number diagnostic

where

$$ identifies Overlay Loader as the message source.

oplb is the operational label of the device or file on which the error occurred.

device type and number identify the device.

diagnostic is an error diagnostic (listed below) corresponding to an I/O completion code.[t]

The following diagnostics may occur:

UNRECOVERABLE I/O ERROR

CALLING SEQUENCE ERROR

INVALID OPERATIONAL LABEL

OL = 0, OR OPERAT MEANINGLESS

ILLEGAL END OF FILE

END OF TAPE

INCORRECT RECORD LENGTH

ILLEGAL BUFFERING

WRITE PROTECTED

BEGINNING OF TAPE

ILLEGAL RAD SEQUENCE

BLOCKING BUFFER UNAVAILABLE

An example of the I/O abort message is given below:

$$ BI MTD0 END OF TAPE

!!BKGD IO ABORT, LOC 3F4C

where

BI is the oplb.

MTD0 is the device name and number.

END OF TAPE is the diagnostic.

3F4C is the I/O abort location.

LOADER ERROR MESSAGES

The Overlay Loader loading error messages are listed in Table D-2.

The type of message is indicated as follows:

A Error causing an Overlay Loader abort, i.e., error message is followed by an abort message.

R Error or condition causing an operator response to be solicited, i.e., the message is followed by an RBM "!!BEGIN WAIT" message.

W Warning message only; loading proceeds.

[t] See Table 10, "I/O Completion Codes", in Chapter 4.

Table D-2. Loader Error Messages

Message	Type	Meaning
$$ LIBSYM UNDEFINED[t] (OLOAD only)	A	There was no file entry on the system Data area of the RAD or disk pack for the LIBSYM table. Overlay Loader aborts with code PL.
$$ ERR BU	W	Sufficient blocking buffer space unavailable. Severity level is set.
$$ ERR CC	R	A control command card has a format or parameter error. An S key-in causes the next control command to be read in from CC. This may be a corrected command to replace the one in error.[tt]
$$ ERR CS	R	There was a checksum error on a binary record. An S key-in causes the record to be reread.[tt]
$$ ERR CO	W	Foreground COMMON, based below root, overlaps root. Warning only, no severity level set.
$$ ERR C1	W	The Loader has encountered COMMON allocation in the root of a non-resident foreground program with the R option specified but without cmn specified on the !OLOAD command. The R specification is ignored and COMMON base is set =K:BACKP minus the size of COMMON.

Table D-2. Loader Error Messages (cont.)

Message	Type	Meaning
$$ ERR IB	R	Illegal binary format (that is, the first word was not 'FF' or '9F') was detected. An S key-in causes the record to be reread.[tt]
$$ ERR ID	R	The indent on the binary module just loaded does not compare with the indent specified on the !$LD command. On an S key-in, the Loader accepts the binary module as is and continues processing.
$$ ERR IS	R or A	Control commands were improperly sequenced in the control command stack. An S key-in causes the next control command to be read. However, if the sequence error was due to a SEG command, the Loader aborts.[tt]
$$ ERR RC	W	Trailing reserve overlapped COMMON; no error severity level is set.
$$ ERR SQ	R	There was an incorrect sequence number on a binary record. An S key-in causes the record to be reread.[tt]
$$ ERR TA	W	No transfer address was encountered in the loading of the root program portion. The Loader sets a default transfer address as the first word of the program and generated an error severity level of one.
$$ ERR UR[t]	W	There were unsatisfied references in the path.
$$ TOO MANY DEFS[t] (OLOAD only)	A	There were more DEFs in the Public Library than were allocated at system generation. Overlay Loader aborts with code PL.
$$ PUBLIB NOT LOADED (OLOAD only)	A	Severity level greater than zero was encountered or generated during Public Library loading. Overlay Loader aborts with code PL.
$$ ERR US	W	A symbol table entry was not recognized.
$$ ERR XL	W	Exloc of program is outside the appropriate area.

[t]This message (OLOAD only) may be written on DO during writing of the Public Library, LIBSYM, or TVECT table onto the disk. If the alarm occurs, the Public Library was not completely written and will have to be reloaded after the error is corrected.

[tt]The Loader does not reposition the record for rereading. If paper tape or cards are repositioned, the record is reread; if they are not repositioned, the next record is read. If the record is on disk or magnetic tape, the Monitor I/O error recovery procedures positions to the beginning of the next record. However, the WAIT permits the taking of dumps, etc., before changing the environment.

LOADER ABORT CODES

Table D-3 lists the abort codes specific to conditions detected by the Overlay Loader during the loading process. The codes appear in the standard abort message of the form

```
!!BKG  xx  ABORT  LOC  zzzz
```

where

xx is the abort code.

zzzz is the location at which the abort occurred (if significant).

RAD EDITOR MESSAGES AND ABORT CODE

The RAD Editor error and warning messages are listed in Table D-4. The type of message is indicated as follows:

- A Error causing a RAD Editor abort, i.e., error message is followed by an abort message.
- R Error causing an operator response to be solicited. (usually only if attend mode is in effect, abort otherwise), i.e., error message is followed by an RBM "!!BEGIN WAIT" message.
- W Warning message only; RAD editing proceeds.
- AC Error causing RAD Editor to abort the current command processing; reads next command.

Table D-3. Overlay Loader Abort Codes

Code	Meaning
A1	Error in accessing the RBMSYM file.
A2	Error in accessing the LIBSYM file.
A3	Error in accessing the EBCDIC library file.
A4	Error in accessing the DEFREF library file.
A5	Error in accessing the MODIR library file.
A6	No blocking buffer is available for the RBMID file.
A8	Error in accessing the TVECT file.
	These codes are frequently caused by an insufficient allocation of RAD Device File Numbers at SYSGEN.
A9	Error in closing the RBMID file.
BB	Cannot assign blocking buffer for input.
CM†	A COMMON displacement or size larger than that stipulated on the !OLOAD command or in a start item was detected. (Background abort only.)
DS††	The same identifier was used to name two different segments.
EF††	An illegal end-of-file was detected.
EL	Excessive Length. The run-time size of the program being loaded has exceeded the specified or default limit (see Chapter 7, Table 19).
IT	An illegal item type was detected.
LI	The library files cannot be loaded because of incorrect construction of the library.
L2†	Labeled COMMON data (subtype 2) is for a block outside the current segment.
L3††	The number of Labeled COMMON indicies allowable per module has been exceeded (currently limited to 40).
L4††	Block size prescribed (subtype 0) is greater than that already allocated.
L5††	Labeled COMMON symbol is defined as a program symbol within the current path.
L6†	Labeled COMMON data from a Library Module (root) is intended for a block allocated in the program section of the root.
L8††	An external DEF was encountered with the same label as a prior labeled COMMON block.
LS	Library search overflow. The number of unique library definitions and references along a program path exceed 300.
On	An Overlay Loader function that prevents proceeding has occurred. The number of the overlay in which the malfunction occurred is indicated by n.
PL	OLOAD was unable to write the Public Library, the LIBSYM, or the TVECT files onto the RAD.
RL	Root of excessive length.
RS	Overlay Loader unable to correctly read the RBMSYM file from the SD area.
SA††	Not enough segments were allocated for the task. The segments parameter of the !OLOAD command should be larger.

Table D-3. Overlay Loader Abort Codes (cont.)

Code	Meaning
SD	Next segment of the Overlay Loader cannot be loaded.
SE	Input ROM had an error severity level greater than zero.
SG[tt]	Format or parameter error was detected on a !$SEG command.
SL	The length of a segment was excessive, (see !$ROOT and !$SEG commands for maximum segment size).
TO[tt]	There was a table overflow. Decrease the size of the program (OLOAD only) or reduce the number of external symbols.
UN[tt]	The number (on the !$SEG card) of the segment to which this one is attached has not been defined.

[t]Loading will continue until terminated but the load program will not be generated and exit will be through M:ABORT.

[tt]Loading will be terminated and, if a map has been requested, it will follow to the point of termination, after which the exit will be through M:ABORT.

Table D-4. RAD Editor Error and Warning Messages

Message	Type	Meaning
## ASSIGN ERR: area, filename	A	The RAD Editor was unable to assign an operational label to a filename because the number of available RAD or disk pack device-file numbers is insufficient or because the specified file does not exist.
## BAD IDENT	A	The object module on BI does not have the same "identification" in the start module item.
##BTL DOES NOT EXIST ON DEVICE	A	The disk pack does not have a bad track list written in sector 2 which is necessary for !#GDTRACK or !#BDTRACK processing.
##BTL OVERFLOW	W	There are more flawed tracks on the disk pack than there are available alternate tracks.
## CALL SEQ ERR oplb	A	A calling sequence error occurred for input/output on the device having the operational label oplb.
## CAN'T FIND area, filename	W	An attempt was made to save, clear, truncate, or delete a file whose name does not exist in the specified area, or the specified area does not exist.
## CHCK WRITE ERR	A	A check write error occurred (that is, data recorded on the disk could not be verified).
## CKSM ERR	R or A	The last record in the object module being read from BI has a checksum error. If the job is in attend mode, operator response is solicited; an operator response of S causes the Editor to read the next record from BI.
## CLEARING ## DELETING ## SQUEEZING ## TRUNCATING } area	R	These messages (followed by !!BEGIN WAIT) are output whenever the indicated operation is started. A key-in of S allows the operation to proceed.
## CORE OVERFLOW	A	The last command cannot be processed for lack of background space.
## DONE	W	Message is output when the operation is completed.

Table D-4. RAD Editor Error and Warning Messages (cont.)

Message	Type	Meaning
## DUP IDENT	A	The last object module read from BI cannot be added to the library with a !#LADD command because it is already in the library.
## DUPLICATE: area, filename	W	An attempt was made to add a file whose name already exists for this area.
## EDIT ERR	A	File directory data on the disk has been rendered invalid.
## EMPTY oplb	R	The device assigned to the operational label is in manual mode.
## EOF oplb	A	An unexpected end-of-file was encountered on the device having the operational label oplb.
## EOF READ FILE	W	An EOF has been encountered on the input file. Copying will continue until EOT on the read file or EOT on the write file is encountered.
## EOT oplb	A	An unexpected end-of-tape was encountered on the device having the operational label oplb.
## EOT WRITE FILE	W	An unexpected EOT occurred on the file currently receiving data. This is a warning to the user that the output file is smaller than the input file (as in !#FCOPY) but that the data already written is correct. The RAD Editor reads the next command.
## {FG / SY} PROTECTED: area, filename	R	The specified area or filename has an SY or FG write protect code and an SY key-in is not in effect. This message will be followed by !BEGIN WAIT.
## FORMAT CONFLICT: area, filename	W	The filename being restored to the area conflicts in format or record size with the existing filename in the area.
## xxxx HAS ALT	W	An alternate track already exists in the bad track list for track xxxx.
## IDENT NOT FOUND	W	The identification in start module item is blank, or there is no object module on BI.
## ILLEG BIN	A	An illegal binary record (first byte not X'FF' or X'9F') has been read in an object module on BI. RAD Editor aborts.
## INV CTRL	W	Control command is invalid. It cannot be recognized by RAD Editor or has incorrect syntax.
## INV I/O OP oplb	A	An invalid input/output operation was attempted on the device having the operational label oplb.
## LENGTH ERR oplb	A	A record of incorrect length was read from or written on the device having the operational label oplb.
## LOAD ERR	A	The required RAD Editor overlay cannot be loaded.
## LOC pppp ERR:0001	A	During the !#GDTRACK-!#BDTRACK processing, the device number specified was 1) not found in the system tables or, 2) was found to be a RAD rather than a disk pack.
## LOC pppp ERR:0002	A	An end-of-tape was detected while writing the bad track list on sector 2.
## LOC pppp ERR:0003	A	An end-of-tape was detected while reading the bad track list on sector 2.
## LOC pppp ERR:0010	A	An error was detected while assigning the operational label X1 to the device number specified in the !#GDTRACK or !#BDTRACK command.

Table D-4. RAD Editor Error and Warning Messages (cont.)

Message	Type	Meaning
## LOC pppp ERR:0100	AC	A !#GDTRACK or !#BDTRACK command requires a minimum of two fields (e.g., 0 < DN ≤ FF plus a track number or 'ALL').
## LOC pppp ERR:0102	A	The device number field on the !#GDTRACK or !#BDTRACK command is a null field or zero.
## LOC pppp ERR:0110	AC	The track number on a !#GDTRACK or !#BDTRACK command must be numeric and 0 < track # ≤ maximum track number for the device.
## LOC pppp ERR:0111	AC	User tried to !#GDTRACK or !#BDTRACK track zero.
## LOC pppp ERR:0120	A	An I/O error occurred during the write headers on the disk pack.
## LOC pppp ERR:0130	A	A RAD device number was used on a !#GDTRACK or !#BDTRACK command.
## LOC pppp ERR:0140	AC	User tried to create a bad track list on an inappropriate disk pack (Model 7242 and 7246).
## LOC pppp ERR:0150	A	An I/O error occurred during the read headers on the disk pack.
## LOC pppp ERR:0170	AC	User tried to use command !#BDTRACK +dn,ALL on an inappropriate disk pack (Model 7251 or 7252).
## LOC pppp ERR:0200	A	RAD Editor cannot find the device number specified on the !#INITIALIZE command in the system tables.
## LOC pppp ERR:0210	A	The device number on the !#INITIALIZE command specifies a RAD.
## LOC pppp ERR:0230	A	The bad track list for the specified device number does not exist in the system tables.
## LOC pppp ERR:0260	A	No device format exists for the specified device number.
## LOC pppp ERR:0261	A	An undefined device format code was found in the system tables.
## LOC pppp ERR:0300	A	During the !#DPCOPY processing, no device format code was found for the specified disk pack.
## LOC pppp ERR:0310	A	The device number specified in the !#DPCOPY command was not found in the I/O Control Subtable.
## MAX TRACK # EXCEEDED	W	User has tried to !#GDTRACK or !#BDTRACK using a track that does not exist on the disk pack.
## NO ALTERNATE	W	An alternate track is not available for execution of the !#BDTRACK command.
## NO BLOCK oplb	A	No blocking buffer is available for the file assigned to the operational label oplb.
## NO GD/BD TRACK ON TRACK 0	W	User cannot !#GDTRACK or !#BDTRACK track zero.
## OVERFLOW: area, filename	W	Allocation of the amount of storage indicated by the file parameter on the !#ADD command or restoration of a file not currently allocated would cause the permanent area to overflow, or a library file has overflowed during execution of a !#LADD command.
## PARAM ERR	W	Control command has a parameter error. A parameter has incorrect content, has been omitted, or is not consistent with the other parameters.

Table D-4. RAD Editor Error and Warning Messages (cont.)

Message	Type	Meaning
## OPEN FILE, NO CHANGE: area, file	W	The file has an operational label assigned to it when a !#DELETE, !#TRUNCATE, !#CLEAR or !#SQUEEZE command is executed and the position of the file changed. The file must be reassigned before it is used by another processor.
## SAVE TAPE NOT AT LOAD POINT	R	The save tape was not at load point when the !#SAVE command was encountered and execution commenced.
## SEQ ERR	Ror A	The last record in the object module being read from BI has a sequence error.
## SZ ERR	A	The object module on BI cannot be placed in the library because it has more than 61 external definitions and references.
## TRACK ZERO BAD	W	During construction of a bad track list, track zero is found to be flawed.
## TRK xxxx NOT IN BTL	W	User has tried to !#GDTRACK a track that does not exist in the bad track list.
## TRUNCATED OPEN FILE: area, filename	W	The user truncated an active file.
## UNRECOVER I/O oplb	A	An irrecoverable I/O error occurred on the device assigned to the operational label oplb.
## WRITE PRO oplb	A	The file name assigned to the operational label oplb is SY or FG write protected and an SY key-in is not in effect.
## DO NOT ABORT DURING SQUEEZE	W	SQUEEZE in process.

For RAD Editor aborts initiated by the RAD Editor itself (i.e., not due to an X key-in by the operator), the following abort message is issued:

```
!!BKG  RE  ABORT  LOC  zzzz
```

where

RE is the RAD Editor abort code.

zzzz is the location at which the abort occurred (if significant).

UTILITY SUBSYSTEM MESSAGES AND ABORT CODE

UTILITY ERROR MESSAGES

The error messages issued by the Utility subsystem are listed in Table D-5. If attend mode is in effect, most of these messages will be followed by a !!BEGIN WAIT message on the OC device (abort otherwise). Only a few of the messages are followed by an unconditional abort (code UT) as indicated in Table D-5.

UTILITY SUBSYSTEM ABORT CODE

Aborts of Utility Subsystem processing are indicated by the following form of abort message.

```
!!BKG  UT  ABORT  LOC  zzzz
```

where

UT is the UTILITY abort code.

zzzz is the location at which the abort occurred (if significant).

Table D-5. Utility Error Messages

Message	Meaning
** BOT oplb, device	An attempt has been made to backspace over the magnetic-tape load point or the beginning of a disk file, i.e., BOT was encountered before the required number of records or files had been passed.
** CAL SEQ ERR	The Utility Executive has encountered a calling sequence error on a return from M:READ/M:WRITE. One reason may be an attempt to copy a record with an odd byte count onto disk (may occur with BCD 7-track tapes). See M:READ status returns in Chapter 4 of this manual.
** CKSM ERR oplb,device	A checksum error was detected on a record read from UI or BI.
** CORE OVFLO	The available memory area used for prestoring commands or storing input records (when the CORE option on the !UTILITY COPY command is used) has overflowed. The Utility program aborts.
** DELETE ERR	No UI card images were found in the block to be deleted (for !*DELETE and !*SUPPRESS commands). Message on DO only unless in attend mode.
** DEOF oplb,device	Two consecutive file marks were encountered before the required number of records or files had been passed, i.e., skipped, compared, etc., or before the program to be updated had been encountered.
** EMPTY oplb,device	Manual intervention is required (the device is in the manual mode or no device is recognized).
** EOF oplb,device	An unexpected tape mark, end-of-file (disk), or !EOD has been read from magnetic tape, cards, paper tape, keyboard/printer, or disk file, e.g., before a required number of records was passed.
** EOT oplb,device	The end-of-tape or end of disk file was encountered before the required number of records or files had been passed.
** ERR AREA	An invalid RAD area name has been used.
** ERR FRGD	An attempt has been made to assign a background operational label to a foreground operational label, device-file number, or RAD file.
** ERR OPLB1	The operational label to be assigned is invalid.
** ERR OPLB2	An attempt has been made to assign one operational label to an invalid or undefined operational label or RAD file.
** IL RAD SEQ oplb,device	The operational label was invalidly assigned to a random-access or compressed EBCDIC disk file, or an attempt was made to skip, read, or write more than one disk file.
** ILLEG BIN oplb,device	The first byte of a record read from UI or BI did not contain X'FF' or X'9F'.
** INV CTRL	A !*MODIFY control command was interpreted from SI when the Record Editor was not in the modify mode.
** INV OPLB oplb,device	The operational label is not valid. The "oplb,device" portion of the message may contain invalid data if input/output is attempted for an operational label not recognized by the Monitor.
** INV I/O OP oplb,device	An input/output operation is not meaningful for the requested device.

Table D-5. Utility Error Messages (cont.)

Message	Meaning
** I/O ERR oplb,device	The input/output calling sequence is in error, incorrect length is specified, or no input/output is pending for a check operation.
** LD INPUT UI,device	The modify mode was entered and updating is to be performed. The operator responds by mounting the tape to be input and keying-in an S response on OC to continue.
** LD LIST UI,device	Both SI and UI are assigned to the same device. The operator responds by mounting the tape to be listed and changes the state of the device.
** NO SPARES	An attempt has been made to define a new background operational label but no room is available in the corresponding table.
** NO name oplb,device	Two consecutive !EODs or tape marks on UI, or one !EOD or tape mark on BI were encountered during the editing process before the desired number of modules had been copied (where "name" is the program name not found).
** NO name UI,device	Two consecutive !EODs or file marks (one end-of-file for a sequential-access RAD file) are read from UI before the Object Module Editor has inserted, replaced, or deleted all requested modules.
** OPLB TABLE OVFL	An attempt has been made to assign or input more than eight operational labels. Only the first eight unique labels on an !*OPLB card will be entered in the operational label table.
** PARAM ERR	Case 1. Update data from SI contains an illegal sequence number; that is, a nonnumeric character. An error alarm is also listed on LO. Case 2. A necessary control command parameter was omitted, or was of invalid form (e.g., oplb), or was greater than 32,767. Case 3. The ident parameter (on an !*IDENT card) is greater than 6, the sequence number parameter is less than 2, or the sum of the two parameters is greater than 8.
** PRE ERR	The !*PRESTORE command did not follow immediately after the !*UTILITY command.
** PRE OVFLO	The RAD prestore file on X5 has overflowed. The Utility program aborts.
** SEQ ERR oplb,device	A sequence error was detected in a record read from SI, UI, or BI. An error alarm may be listed on LO also. (Message occurs on OC only if attend mode is in effect.)
** UNRECOV I/O oplb,device	An irrecoverable input/output error has occurred after the maximum number of retries has been unsuccessfully attempted.
** UNRECOV I/O UI,device	An irrecoverable read error has occurred on UI. The partial card image input and the message "UI IGNORED RECORD FOLLOWS xxxxxxxx" (when xxxxxxxx is the previous nonblank UI ident and/or sequence number) is output on LO.
** UNRECOV I/O UO,device	An irrecoverable write error has occurred on UO. The card image to be output, and the message "UO RECORD OMITTED" or "UO FILE MARK OMITTED", are output on LO.

Table D-5. Utility Error Messages (cont.)

Message	Meaning
** VERIFY ERR oplb,device	An error has been found by the verification process. When a verification error occurs, the COPY routine terminates execution of the !*VERIFY command for that device, but continues verification on other input devices. If an error is detected on every input device, the VERIFY function is terminated.
** WRITE PRO oplb,device	An attempt has been made to write on a write-protected magnetic tape or RAD file.

APPENDIX E. USASCII-8 TO EBCDIC-8 CORRESPONDENCE

COL / ROW	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	K0	K16	SP	&	–	N26	N35	N24	N49	N56	{	}	\	0
	0/0	1/0	8/0	9/0	2/0	2/6	2/3	11/12	12/3	12/10	13/1	13/8	7/11	7/13	5/12	3/0
1	SOH	DC1	K1	K17	N0	N9	/	N27	a	j	~	N57	A	J	K31	1
	0/1	1/1	8/1	9/1	10/0	10/9	2/15	11/11	6/1	6/10	7/14	13/9	4/1	4/10	9/15	3/1
2	STX	DC2	K2	SYN	N1	N10	N18	N28	b	k	s	N58	B	K	S	2
	0/2	1/2	8/2	1/6	10/1	10/10	11/2	11/12	6/2	6/11	7/3	13/10	4/2	4/11	5/3	3/2
3	EXT	DC3	K3	K19	N2	N11	N19	N29	c	l	t	N59	C	L	T	3
	0/3	1/3	8/3	9/3	10/2	10/11	11/3	11/13	6/3	6/12	7/4	13/11	4/3	4/12	5/4	3/3
4	K28	K29	K4	K20	N3	N12	N20	N30	d	m	u	N60	D	M	U	4
	9/12	9/13	8/4	9/4	10/3	10/12	11/4	11/14	6/4	6/13	7/5	13/12	4/4	4/13	5/5	3/4
5	HT	K5	LF	K21	N4	N13	N21	N31	e °	n	v	N61	E	N	V	5
	0/9	8/5	0/10	9/5	10/4	10/13	11/5	11/15	6/5	6/14	7/6	13/13	4/5	4/14	5/6	3/5
6	K6	BS	ETB	K22	N5	N14	N22	N32	f	o	w	N62	F	O	W	6
	8/6	0/8	1/7	9/6	10/5	10/14	11/6	12/0	6/6	6/15	7/7	13/14	4/6	4/15	5/7	3/6
7	DEL	K7	ESC	EOT	N6	N15	N23	N33	g	p	x	N63	G	P	X	7
	7/15	8/7	1/11	0/4	10/6	10/15	11/7	12/1	6/7	7/0	7/8	13/15	4/7	5/0	5/8	3/7
8	K23	CAN	K8	K24	N7	N16	N24	N34	h	q	y	G0	H	Q	Y	8
	9/7	1/8	8/8	9/8	10/7	11/0	11/8	12/2	6/8	7/1	7/9	14/0	4/8	5/1	5/9	3/8
9	K13	EM	K9	K25	N8	N17	N25	\	i	r	z	G1	I	R	Z	9
	8/13	1/9	8/9	9/9	10/8	11/1	11/9	6/0	6/9	7/2	7/10	14/1	4/9	5/2	5/10	3/9
A	K14	K18	K10	K26	[	]	¦	:	N36	N43	N50	G2	G8	G14	G20	G26
	8/14	9/2	8/10	9/10	5/11	5/13	7/12	3/10	12/4	12/11	13/2	14/2	14/8	14/14	15/4	15/10
B	VT	K15	K11	K27	.	$	,	#	N37	N44	N51	G3	G9	G15	G21	G27
	0/11	8/15	8/11	9/11	2/14	2/4	2/12	2/3	12/5	12/12	13/3	14/3	14/9	14/15	15/5	15/11
C	FF	FS	K12	DC4	<	*	%	@	N38	N45	N52	G4	G10	G16	G22	G28
	0/12	1/12	8/12	1/4	3/12	2/10	2/5	4/0	12/6	12/13	13/4	14/4	14/10	15/0	15/6	15/12
D	CR	GS	ENQ	NAK	(	)	-	'	N39	N46	N53	G5	G11	G17	G23	G29
	0/13	1/13	0/5	1/5	2/8	2/9	5/15	2/7	12/7	12/14	13/5	14/5	14/11	15/1	15/7	15/13
E	SO	RS	ACK	K30	+	;	>	=	N40	N47	N54	G6	G12	G18	G24	G30
	0/14	1/14	0/6	9/14	2/11	3/11	3/14	3/13	12/8	12/15	13/6	14/6	14/12	15/2	15/8	15/14
F	SI	US	BEL	SUB	|	¬	?	"	N41	N48	N55	G7	G13	G19	G25	EO
	0/15	1/15	0/7	1/10	2/1	5/14	3/15	2/2	12/9	13/0	13/7	14/7	14/13	15/3	15/9	15/15

APPENDIX F. LINE PRINTER VFCs (WRITE BINARY)

Pseudo VFC	Print with Format Definition	Real VFC	Print Order	Data Chained to Text (Yes/No)	Printer Model
X'60'	Print, suppress upspace	X'60'	PF	Yes	A, B, C
X'80'	Print, suppress upspace	X'60'	PF	Yes	A, B, C
X'81'	Print, then space 1 line	X'C0'	PF	Yes	A, B, C
X'82'-X'8F'	Print, then space n lines (2-15)	1) X'60' 2) X'C0'+n	PF F	Yes No	A, B, C
X'90'-X'9F'	Print, then skip to channel n	1) X'60' 2) X'F0'+n	PF F	Yes No	A, B, C
X'A0'-X'AF'	Space n lines, print and inhibit upspace	1) X'C0'+n 2) X'60'	F PF	No Yes	A
		X'E0'+n	PF	Yes	B, C
X'B0'-X'BF'	Skip to channel n, print and inhibit upspace	1) X'F0'+n 2) X'60'	F PF	No Yes	A
		X'D0'+n	PF	Yes	B, C
X'C0'-X'CF'	Space n lines, print and upspace	X'C0'+n	PF	Yes	A, B, C
X'D0'-X'DF'	Skip to channel n, print and inhibit upspace	1) X'F0'+n 2) X'60'	F PF	No Yes	A
		X'D0'+n	PF	Yes	B, C
X'E0'-X'EF'	Space n lines, print and inhibit upspace	1) X'C0'+n 2) X'60'	F PF	No Yes	A
		X'E0'+n	PF	Yes	B, C
X'F0'-X'FF'	Skip to channel n, print and upspace	X'F0'+n	PF	Yes	A, B, C

Notes:
- PF - Print with format
- F - Format
- A - Printer models 3451, 7440, 7445
- B - Printer models 7441, 7442, 7446, 3461, 3463, 3464, 3465, 3466
- C - Printer model 7450
- n - Number of lines to skip or channel number. N is limited by line printer capabilities (e.g., a skip to channel > 1 for the 7450 line printer will result in a skip to channel 1).

Invalid VFCs result in a single space((X'CO') operation.

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.

L

M

N

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

T

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.

U

V

W

X

Z

Corporation
)1 South Aviation Boulevard
El Segundo, California 90245

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

☐ Learning ☐ Installing ☐ Sales
☐ Reference ☐ Maintaining ☐ Operating

Is the material presented effectively?

☐ Fully Covered ☐ Well Illustrated ☐ Well Organized ☐ Clear

What is your overall rating of this publication?

☐ Very Good ☐ Fair ☐ Very Poor
☐ Good ☐ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, Please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your Name & Return Address

2190(12/72)

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

Staple

Fold

First Class
Permit No. 229
El Segundo,
California

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

Postage will be paid by

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

Attn: Programming Publications

Fold

XEROX

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511