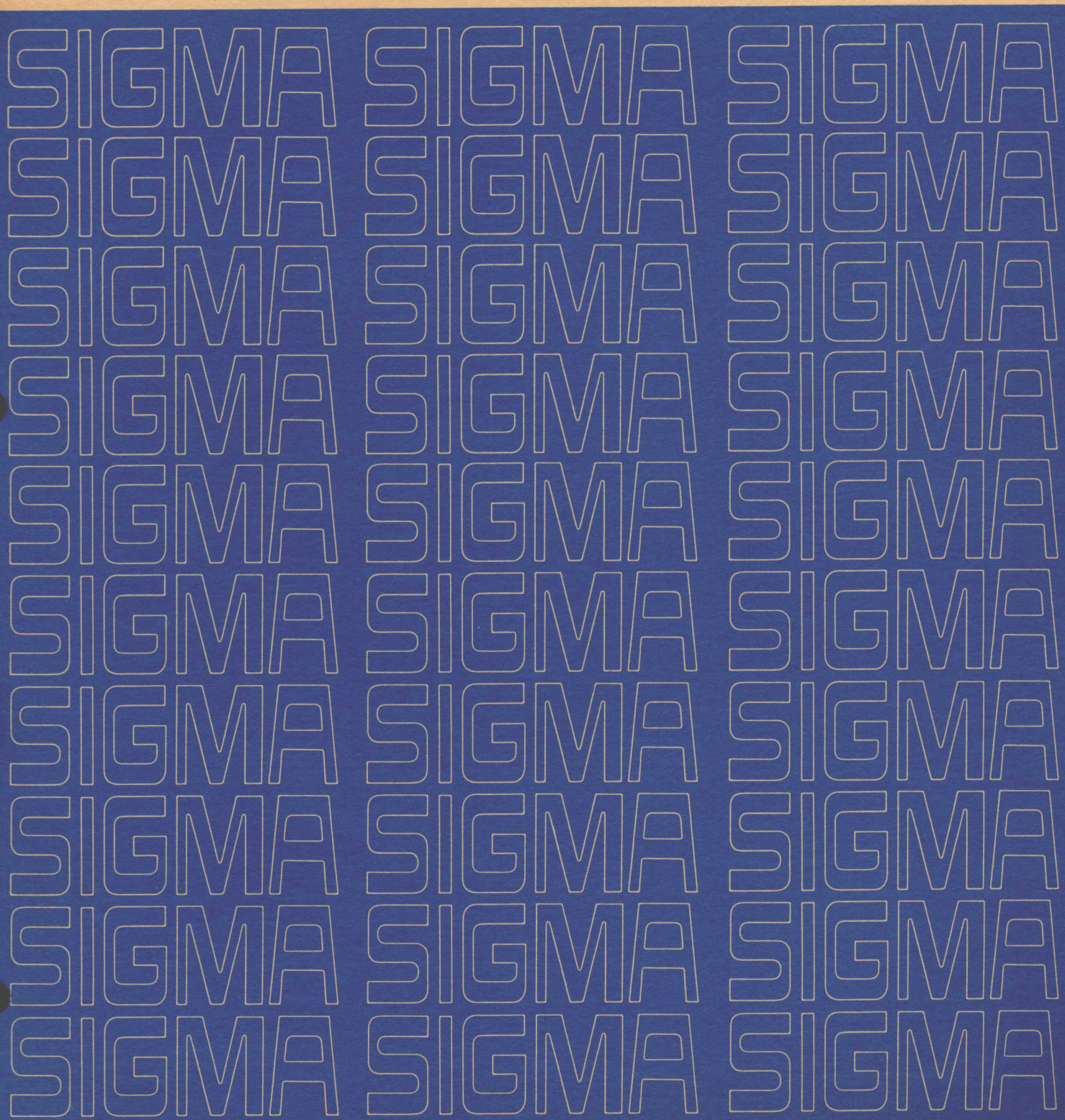
Xerox Data Systems

XEROX

Xerox Real-Time Batch Monitor (RBM)

Sigma 2/3 Computers

Real-Time and Batch Processing Reference Manual



90 10 37E

Xerox Data Systems

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

XEROX

Real-Time Batch Monitor (RBM)

Sigma 2/3 Computers

Real-Time and Batch Processing

Reference Manual

90 10 37E

March 1971

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 Sigma 2/3 computers, Publication Number 90 10 37D (dated August 1970). Primary technical changes made to the text are for the D00 version of RBM. Other changes are a heavy reorganization of the text for improved reader referencing. All technical changes from that of the previous manual are indicated by a vertical line in the margin of the page. Organizational changes in the text are not so indicated.

RELATED PUBLICATIONS

Title	Publication No.
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 Basic FORTRAN and Basic FORTRAN IV/LN, OPS Reference Manual	90 09 67
Xerox FORTRAN Library/System Technical Manual	90 10 36
Xerox Basic FORTRAN IV/OPS Reference Manual	90 15 25
Xerox Extended Symbol/LN OPS Reference Manual	90 10 52

Manual Type Codes: BP - batch processing, LN - language, OPS - operations, RBP - remote batch 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.

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 five 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.

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

device-file number: 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.

dictionary: a directory of names and addresses of files or other catalogs on a random access device that enables the system to locate an item when given only its name.

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.

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: the minimum physical amount of data transferred in a read or write operation from/to 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 control command: any control command used during System Generation to direct the formatting of a Monitor 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 normal sequence of events in the central processor.

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.

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

postmortem dump: 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 absolue 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.

1. INTRODUCTION

RBM CHARACTERISTICS

The 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 B).
- Input/output 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. It 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:

- Sigma 2/3 hardware 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, and (d) optional secondary storage modules.
- Partitioned core memory (see Figure 1) divided into
  - o A protected foreground area reserved for (1) resident real-time foreground programs, (2) a single

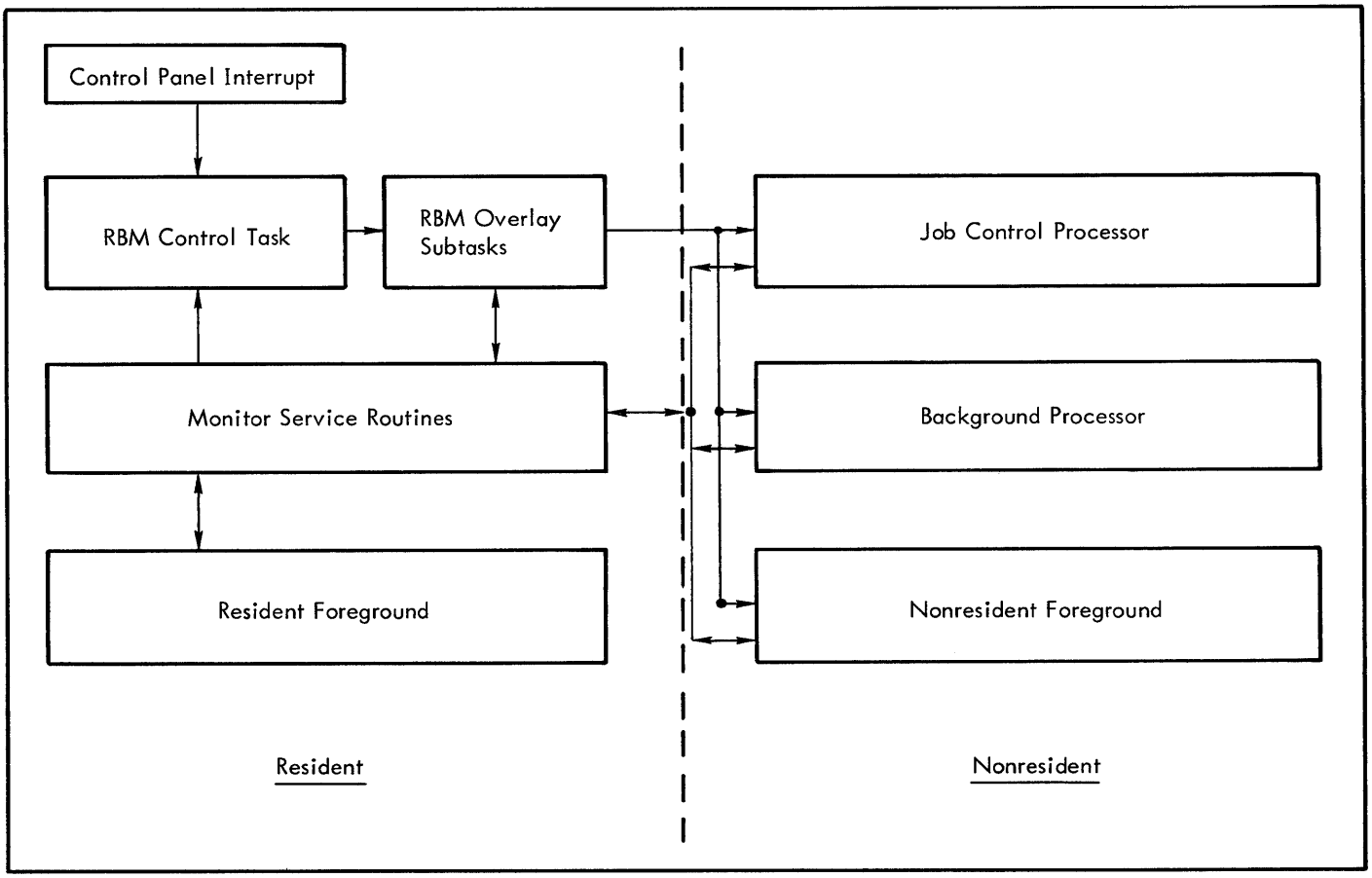


Figure 1. Operating System

nonresident foreground program, (3) Monitor tasks that must respond to high-priority interrupts, (4) Monitor service routines, and (5) optional routines (such as a Public Library) that are used by both foreground and background programs.

- o 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,[t] allocatable into permanent and temporary files. The permanent files contain all of the background RBM processors such as Basic FORTRAN IV, Extended Symbol, RAD Editor, etc., plus RBM itself. They may also contain user data and optional 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.

- Up to 137 (107 for Sigma 3) user foreground tasks that can be connected to 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 B).

- Make available a "mailbox" area of 32 cells of memory for communication between two or more foreground programs.

The interrupt priority sequence (described in detail in the Xerox Sigma 2 and Sigma 3 Computer Reference Manuals) 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.

Monitor 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 the Overlay Loader to link all modules and create the proper overlay structure for execution.

[t] 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 IOCDs 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 in the Sigma 2/3 computers. 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 bookkeeping).
- 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 areas is a convention created primarily to offer a scheme to expedite file management. RAD areas are allocated during system initialization (see RAD Allocation in Chapter 11) and are labeled with two alphanumeric characters, usually from the following list:

SP	UL
SD	BT
SL	CP
UP	Dn
UD	Xn

where n is a hexadecimal digit, and Dn is an area that may contain any data the user desires including program files.

Certain labels of the list above 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 Loader will load library routines.
UP	User Processor area. Contains resident foreground programs, foreground tasks, nonresident programs, semiresident programs, and background programs. Only this area and SP area are searched when a processor is requested.
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.
Xn	Xn areas are similar to Dn 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 a Mount Area key-in (see "Unsolicited Key-Ins" in Chapter 3).

†These areas receive default allocations (see Table 27). 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. The Overlay Loader stores 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. When allocating files, any file defined in an area with a P as the second character of the mnemonic is considered a processor file.

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. The Overlay Loader can load selectively from one or both, in either order of 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. RBM maintains no details on content, addressing, or record size. When allocating files, any file defined in an area with a D as the second character of the area mnemonic is considered a data file.

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, the Overlay Loader 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 the 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 Loader. 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 Loader, see the Overlay Loader chapter.

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

All RBM service routines and Sigma 2/3 system library routines (FORTRAN and mathematics libraries) are reentrant. If an RBM system 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.

REENTRANT ROUTINES

As used in Sigma 2/3 software, "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 Sigma 2/3 priority interrupt system.

ACCOUNTING AND ELAPSED TIME

Background job accounting and provisions to limit the execution time of a background job can be accomplished via Clock 1. (The use of Clock 1, an interrupt device, is optional at SYSGEN initiation.) To correctly calculate the elapsed time for the background, the Monitor M:SAVE routine records the start time of the first interrupting foreground task and triggers the RBM Control Task to calculate the actual foreground run time. By performing this calculation at the priority level of the RBM Control Task, rapid response time for the foreground is maintained.

Clock 1 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 the RBM Control Task will be triggered every 16 seconds to 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 since the start of the job 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 W key-ins or the result of 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 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.
- A !JOB or !FIN command writes out total accumulated times and resets times to zero.
- The ET (elapsed time) represents the total background accumulation since !JOB was encountered. ET is printed out each time CCI is read into the background.

SYSTEM INITIALIZATION AND CREATION

The RBM system creates itself for a particular installation through a nonresident SYSGEN routine. The permanently resident, nonoptional parts of RBM are loaded into low core; next, the RBM initializer is loaded along with the optional RBM routines and the standard input/output definitions and tables.

The user then defines RAD areas, optional routines, the 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. The system is then complete in lower memory.

Once the system is completely defined, routines not needed will be discarded and an absolute rebootable version is punched on a binary output device (optional) and a rebootable version is written onto the RAD. The system initializer is overwritten by the first background program or real-time foreground program loaded just below 12K.

If the system must be restarted later, the rebootable version is loaded from the RAD. A completely new system initialization is necessary only if some of these standard 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.

Protection switches on the 7202, 7204, and 7232 equipment may be used to permanently protect certain areas of the RAD.

HARDWARE REQUIREMENTS

The minimum configuration required and supported by RBM for either a Sigma 2 or Sigma 3 is the following:

- Sigma 2 or Sigma 3 CPU with either Internal IOP or External IOP (Sigma 3 only)
- Memory Parity Interrupt
- Memory Protect Feature
- Hardware Interrupt (for RBM Control Task)
- Core Memory Module (8192 words)
- One Memory Increment (8192 words)
- Keyboard/Printer with Paper Tape Reader/Punch
- RAD Controller
- RAD Storage Unit (0.75 M bytes)

An alternate minimum configuration (for a Sigma 3 CPU with external IOP only) is a Disk Pack Controller and Disk Pack Storage Unit to replace the RAD Controller and RAD storage Unit. Other minimum requirements remain the same.

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

- Disk Packs
- Memory Module
- Memory Increment
- Keyboard/Printer
- Paper Tape Reader/Punch (High-Speed)
- Card Readers
- Card Punches
- RADs
- 9-Track Magnetic Tape
- 7-Track Magnetic Tape
- BCD and Binary Packing Options for 7-Track Magnetic Tape
- Line Printers
- Plotters

RBM SUBSYSTEMS

RBM will support the subsystems and processors described below. All execute in the background area of core memory and the collective set offers maximized utilization of Sigma 2/3 computer capabilities.

LANGUAGE TRANSLATORS

EXTENDED SYMBOL

The Extended Symbol programming language (and assembler) provides upward compatibility with basic Symbol in addition to 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 program load 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.

SERVICE PROGRAMS

OVERLAY LOADER

The Overlay Loader forms 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 Dn 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.

CONCORDANCE

The Concordance program provides the user with a listing of program symbols and all references to these symbols by source line number. Optional control cards permit inclusion or exclusion of specified symbols in local, nonlocal, or operation/directive code sections of the printout. Most of the options of Concordance are available under Extended Symbol.

Omission of optional control cards yields a standard Concordance listing containing all program symbols except standard operation and directive code menmonics.

MISCELLANEOUS

DEBUG

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

COC

The character-oriented communications (COC) handler provides communication between Sigma 2/3 real-time programs and various terminal devices. The COC consists of a controller and from one to eight attached line interface units. The Sigma 2/3 RBM can accommodate one COC. See Chapter 4 and Appendix F for a more complete discussion of the COC handler.

RBM TERMS AND PROCESSES

The following items are either unique to the RBM system or have specific meaning within the RBM context. Terms and processes not defined below are explained in the 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 27). 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 memory parity 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 common 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 only by the number of internal and external interrupts possible 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 up the available 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. Sigma 2/3 hardware and the RBM software completely and absolutely protect resident foreground programs from a background program in terms of I/O and core memory usage. Thus, 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 or 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 is not allowed to 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.

BACKGROUND TASK

A "background task" is an executable version of a single background process that shares the same restrictions as other background jobs relative to foreground priorities and privileges.

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.

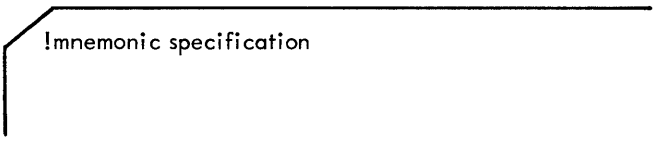
COMPRESSED RAD FILES

EBCDIC character codes do not use all possible bit combinations of an eight-bit byte, and some combinations (X'FC' 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.

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 the control command.

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 service commands.
2. System processor names.
3. User 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 Sigma 2/3 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 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][,$oplb_1$][,$oplb_2$][,$oplb_3$]... [,$oplb_{10}$]

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 require 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 block 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 (and remained fixed) in the DEVICE FILE INFO parameter during SYSGEN (see Table 27).

Standard operational labels can be reassigned to different device-file numbers during SYSGEN or through !ASSIGN and !DEFINE control commands. One table of operational labels is used for the background (see Table 2 below) and another table is used for the foreground. Device unit numbers are also stored in the same two tables as 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†	Execution input (GO)	CR, MT, PT, RD
ID	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†	Overlay (temporary)	RD
PI††	Processor input	RD
PM	Punch RBM	CP, PT, MT

Table 2. Standard Background Operational Labels (cont.)

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

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

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

[ttt]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.

The standard foreground operational labels are as follows:

Operational Label	Explanation of Reference	I/O Device
BO	Binary output	CP,PT,MT
AL	Accounting log	RD

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: some background processors (e.g., Utility) do not allow use of active operational labels assigned to file zero. See Appendix E 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 or background operational labels. (The operator must key in FG before assigning a foreground operational label.)

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

```
!ASSIGN oplb = file number[,F]
```

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).

file number is the device-file number for a physical device in the system (created at SYSGEN).

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.

Form 2 of the !ASSIGN command is

```
!ASSIGN oplb =file name ,area mnemonic [,F]
```

where

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

file name 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 file name. Once the file name 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 to this file name.

area mnemonic specifies the area to search for the file name, usually from the areas listed in Table 4.

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

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

Table 4. RAD Area Mnemonics

Code	Meaning
SP	System Processor area
SD	System Data area
SL,UL	System and User Libraries
UP	User Processor area (user tasks and programs and background processors)
BT	Background Temp area
CP	Checkpoint area
Dn	Data area(s)
UD	User Data area
Xn	Similar to Dn areas but no disk pack verification performed

Form 3 of the !ASSIGN command is

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

where

oplb is as defined above.

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

Examples:

Form 1: !ASSIGN SI = 3

!ASSIGN F:105 = 3

Form 2: !ASSIGN OV =FILE1, UP

Form 3: !ASSIGN LI = BI

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 point 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 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

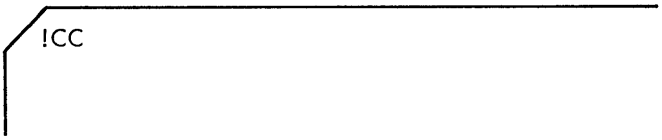
the blocking buffer pool. If a user builds his own TCB, the TCB must begin at the execution location plus the "temp" value specified on the Overlay Loader !$ROOT command.

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.

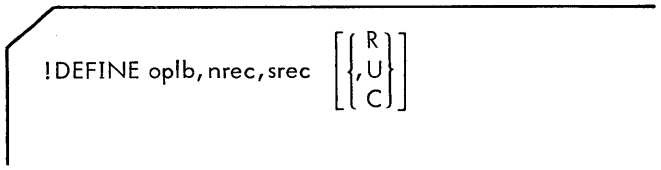
The !C: command does not change the contents of the TCB.

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



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



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.

srec is the logical record size, in bytes.

R defines the file as a random-access file.

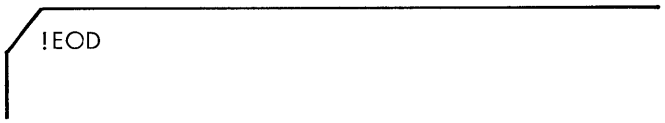
U defines the file as an unblocked file.

C defines the file as a compressed EBCDIC file.

If neither R, U, nor C is specified, the file is defined as a subsequential, noncompressed, blocked file. If R is input, srec is used as the granule size.

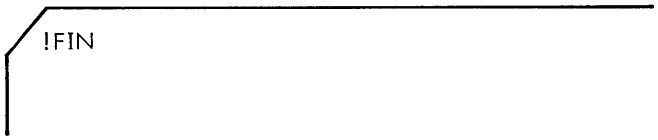
EOD Blocks may be defined in a user's deck by inserting !EOD control commands at the end of each block. When an !EOD command is encountered, the Monitor returns an EOD status (when using the M:READ I/O routine). This is similar to a tape-mark on magnetic tape. Any number of !EOD control commands may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

The form of the !EOD control command is



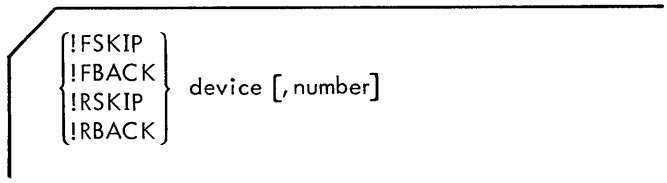
FIN The !FIN control command specifies the end of a stack of jobs. When the !FIN control command is encountered, the Monitor writes it on the listing log to inform the operator that all current jobs have been completed and also writes !!BEGIN IDLE on the OC device. The Monitor then enters the idle state.

The form of the !FIN control command is



FSKIP,FBACK,RSKIP,RBACK The file positioning control commands, !FSKIP and !FBACK, forward or backspace the specified device (magnetic tape or sequential 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 does not apply to compressed RAD files.

The forms of the control command are



where

device is the device indicator of the device to be positioned and is restricted to background devices. The device indicator 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

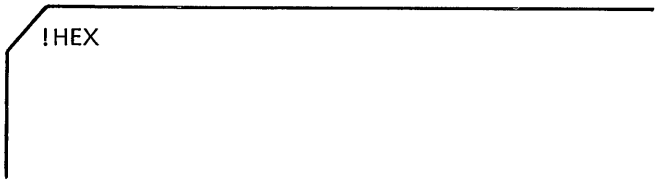
   where n is a decimal integer equal to the device unit number.

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.

HEX The !HEX control command loads patches at execution time for either the Monitor itself or any user program. (See Appendix G for input description.)

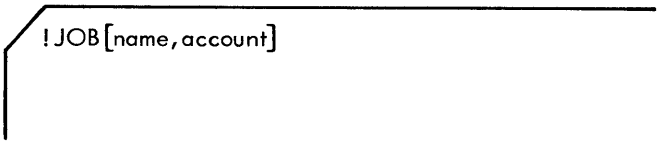
The form of the !HEX control command is



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 standard assignments as defined at System Generation. 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



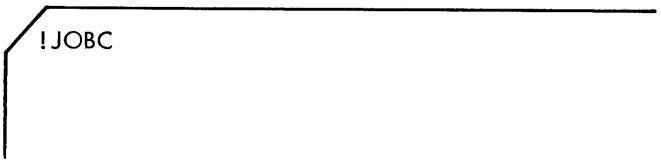
where

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 standard 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



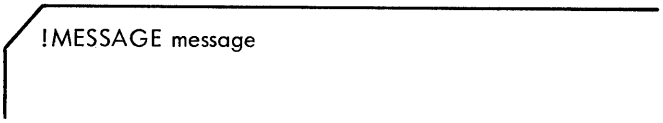
LIMIT The !LIMIT control command is used to set a maximum on the execution time of a background program. This command is effective only if the system has real-time Clock 1 dedicated to the Monitor. If the job exceeds the time limit, the job is aborted (TL) and is terminated with a postmortem dump (if that option was specified).

The form of the !LIMIT control command is

where N is the maximum allowable execution time in minutes ($0 < N < 6000$).

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.

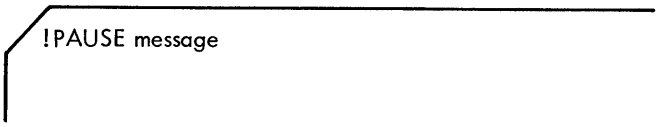
The form of the !MESSAGE control command is



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 if an error occurs during program execution. The dumps are always onto the background DO device in hexadecimal format.

The form of the !PMD command is

```
!PMD [U][,ALL][,from,to]...[from,to]
```

where

U if present, signifies an unconditional dump at the end of the next job step even if there are no errors.

ALL if present, signifies that all of the background memory is to be dumped. If ALL is not present and no limits are specified, only the general registers are dumped.

from specifies the location (decimal or hexadecimal) at which dumping is to begin.

to specifies the last location to be dumped.

Up to four limit-pairs may be specified. The CPU registers are printed in hexadecimal as the first line of the dump regardless of the limits.

PURGE The !PURGE control command is used to output the contents of the accounting file. The output is to background operational label LO in the following format:

MM/DD/YY HRMN NAME ACCOUNT TIME

(MMMM.MM)

An option is provided to clear the accounting file subsequent to this output. In this manner the user could assign background operational label LO to a device such as the card punch or the paper tape punch, and by exercising the "clear" option, could produce a periodic hard copy of the accounting file and clear the accounting file for future use.

The form of the !PURGE control command is

```
!PURGE [C]
```

where C is the directive to clear the accounting file (must be preceded by an unsolicited SY key-in).

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 Sigma 2/3 Standard Object Language. The modules may constitute a complete program, a root, or segments of a program.

The form of the !REL control command is

```
!REL
```

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 sequential RAD file and has no effect on other devices. The operation takes place immediately after the command is interpreted. The command is restricted to background files.

The form of the !REWIND control command is

```
!REWIND device
```

where device is the device indicator (as in !FSKIP) of the device to be rewound.

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

```
!TEMP {S}
      {R}
```

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.

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

The form of the !UNLOAD control command is

```
!UNLOAD device
```

where device is the device indicator (as in !FSKIP) of the file to be rewound off-line.

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

The form of the !WEOF control command is

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

where

device is the device indicator (as in !FSKIP) of 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.

XEQ The !XEQ control command loads the first program 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

System processors on the System Processor area and any user background or foreground program residing in the User Processor area can be called by a processor control command. The commands have the format

```
!processor parameters
```

where

processor is the file name of a processor (see Table 5).

parameters are optional parameters interrupted by each particular processor.

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.

Table 5. RBM System Processors

Name†	Description
!FORTRAN	Basic FORTRAN IV Compiler
!CONCORDA	Concordance Program for Extended Assembler
!OLOAD	Overlay Loader
!UTILITY	Utility
!XSYMBOL	Extended Symbol Assembler
!RADEDIT	RAD Editor

†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 requested processor is read into the background and attains control, this marks the beinning 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.

BO specifies binary output.

CR specifies cross-reference listing.

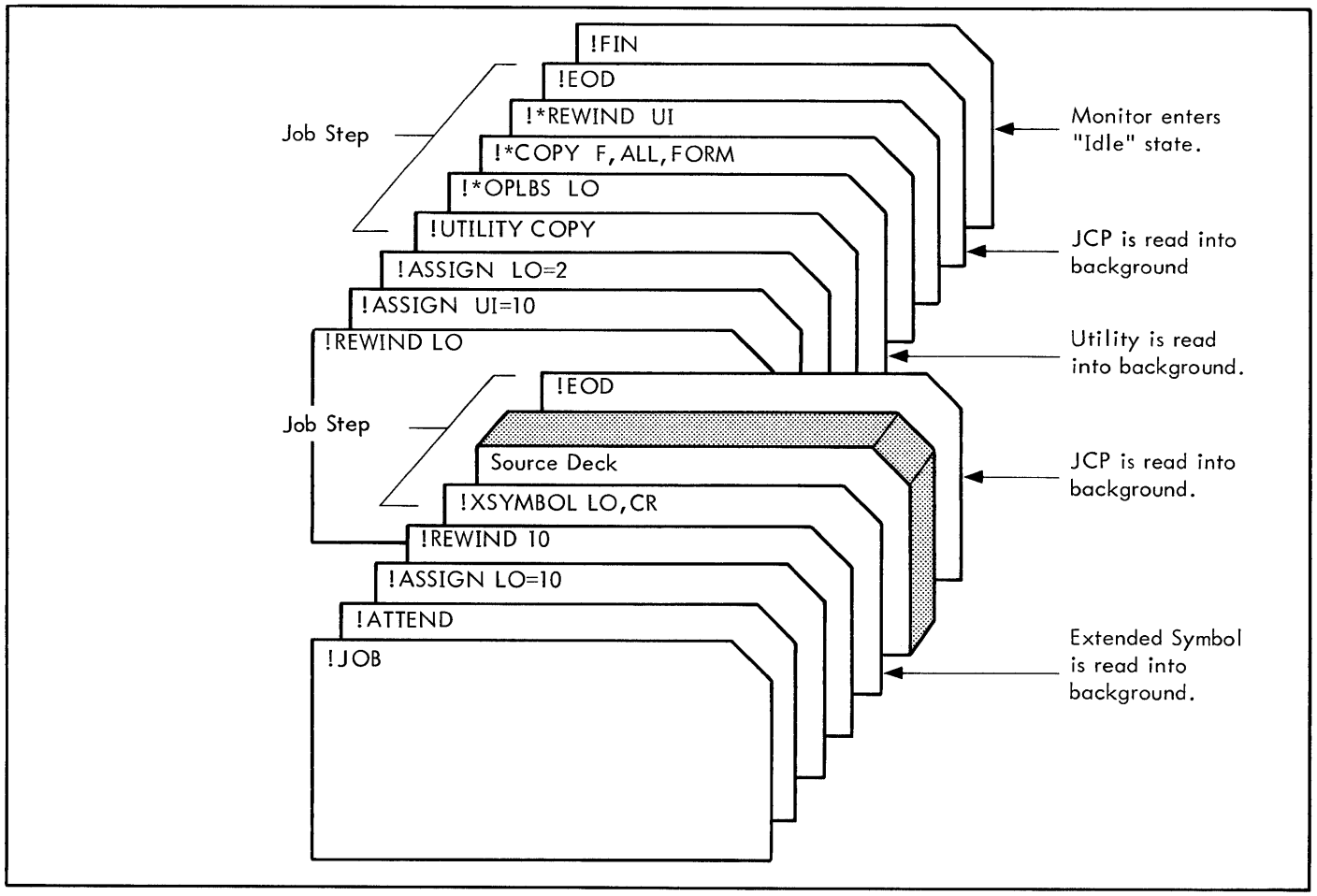


Figure 2. Job Stack Example

DW specifies display warnings.

GO specifies output GO file.

LO specifies list assembly output.

NP specifies no standard procedure input.

NS specifies no summaries.

PP specifies punch standard procedure file.

SL specifies simple literals.

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.

BASIC FORTRAN IV CONTROL COMMAND FORMAT

The Job Control Processor reads and interprets the !FORTRAN control command and loads the Basic 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.

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 device 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 unrecoverable 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 ouput line to LO from an assembly or compilation causes a page eject.

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

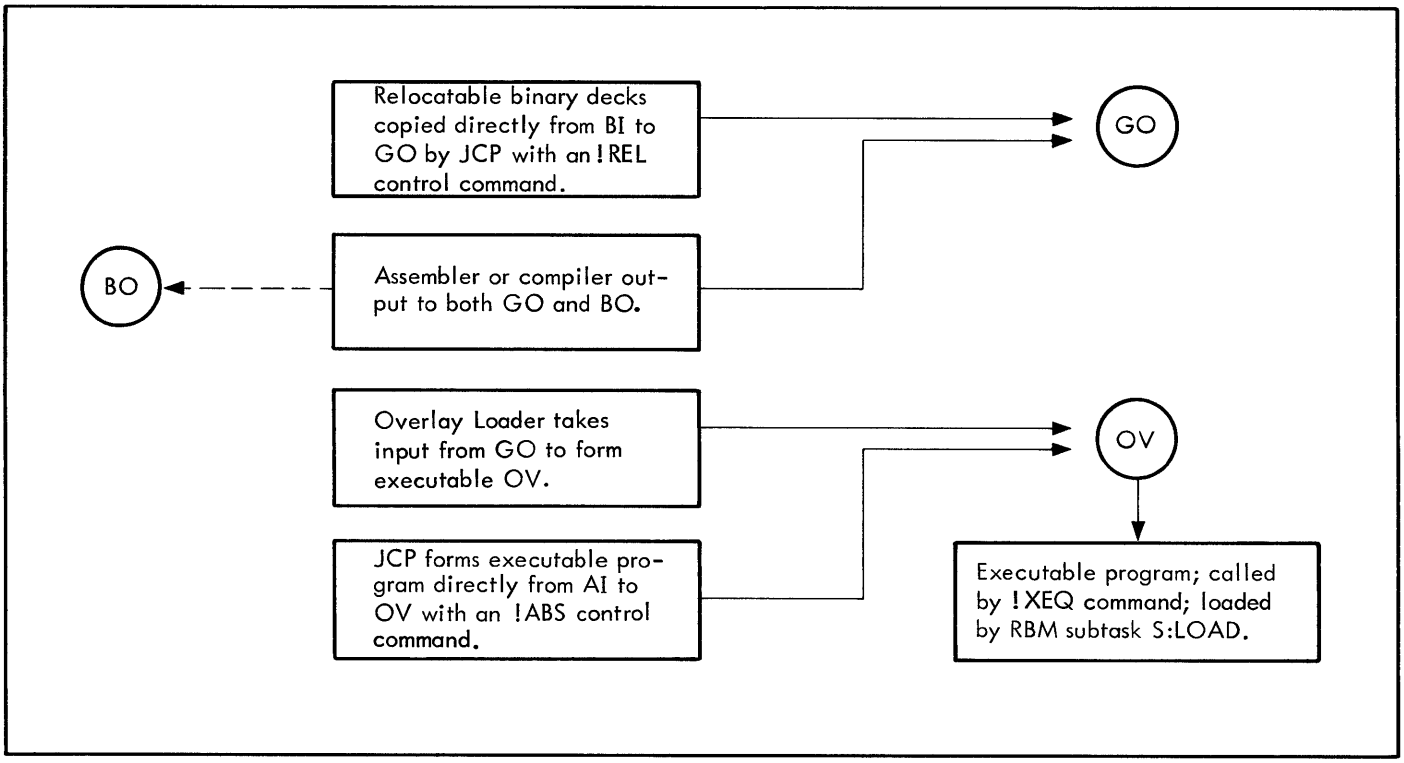


Figure 3. Use of GO and OV Files

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 (!!).

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.

MONITOR MESSAGES

The messages itemized in Table 6 are output on the OC device. They are primarily for background program use but can be used by foreground by specifying standard error recovery and "initiate and wait" in the M:READ, M:WRITE, or M:CTRL calling sequence.

Real-time programs with special requirements can inform the operator of special conditions and wait for an operator response.

Table 6. Monitor Messages

Message	Meaning
!!ABORT CODE xx LOC yyyy	The background job has aborted by reason of code xx (abort codes are defined in Appendix C). If this was a CC abort (control command error), a more explicit reason will be listed on the background DO device. (All abort messages and diagnostics are listed on the DO device.) If the system is operating in an "attend" mode (see Chapter 2), 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 recover and attempt to process the next control command on the CC device. If not operating in the "attend" mode, RBM will not go into a wait state but will perform any required postmortem dumps and immediately begin reading from the standard CC device, skipping all control commands or data cards until a !JOB or !FIN card is found. All control commands are listed on the LL device, with an indication – a '<' preceding the command to show that they are being ignored.
!!AL IO ERROR[t] !!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.

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

Table 6. Monitor Messages (cont.)

Message	Meaning
!!AL OVERFLOW[t] !!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 hard-copy device (e.g., paper tape, card punch). Input of a !PURGE control command specifying the clear option (i.e., !PURGE C) 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 ERRORxx	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.
!!BK RELEASE,dtnn	The specified device has been released for background use.
!!BKG RESTART	Background has been restarted from its point of interruption.
!!CCI	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.
!!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 it is RD, xxxx will be the errored track number.

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

Table 6. 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.
!!dtnn RATE ERR	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	Either the RAD is physically write-protected or a RAD file is logically write-protected. If a RAD file is logically write-protected, an unsolicited key-in of SY will allow RBM to continue. If the background is attempting an invalid operation, it should be aborted.
!!END IDLE	RBM has gone out of the idle state and will begin reading control commands from the CC device. Control commands will be ignored until a !JOB command is input.
!!FG PARITY ERR, TCB=FFFF, LOC=FFFF, A=FFFF, X=FFFF, B=FFFF	A foreground parity has occurred but the specified allowable limit of foreground parity errors has not been reached.
!!FG PARITY ERX, TCB=FFFF, LOC=FFFF, A=FFFF, X=FFFF, B=FFFF	A foreground parity error has occurred. The specified allowable limit of foreground parity error has been reached. ERX indicates that the task has been disabled and terminated.
!!FG 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.
!!FG RESERVE,dtnn	The specified device has been reserved for foreground use.

Table 6. Monitor Messages (cont.)

Message	Meaning
!!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: AREA — The wrong disk pack was mounted for an 'M' key-in. DEVICE — The channel for the device specified was not defined at SYSGEN or this device is not defined. Applies to 'M' and 'BT' key-ins. FIXED — Performing the requested mount would entail undefining more than one other area. OVFLOW — The Master Dictionary, Alternate Track Pool, or IOCS 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 Temp Stack has overflowed.
!!MACH. FAULT: TCB=FFFF, LOC=FFFF, X=FFFF, B=FFFF	Direct I/O to an unrecognized device has been attempted twice and a watchdog timeout has occurred.
!!MACH. FAULX: TCB=FFFF, LOC=FFFF, A=FFFF, X=FFFF, B=FFFF	Direct I/O to an unrecognized device has been attempted twice at the same location. The foreground task subsequently is disabled and terminated.
!!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.
!!POWER ON	The system has experienced a power failure, and the power fail-safe option has been implemented. This message is written at the RBM interrupt level, and consequently, any foreground tasks will have been completed before this messsage is typed. At this point the operator should terminate background, and when foreground is completely idle, he should reboot RBM from the RAD and restart the background. The message is output as soon as RBM has control.

OPERATOR CONTROL

UNSOLICITED KEY-INS

Because of the 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 communication. (Normally, operator communication is on a keyboard/printer.) All background references to the operator output device should be to operational label OC. A frequent method of operator control is in response to a specific request from a foreground or background program. In this case, there is no standard format.

The operator may also desire to exercise control over the background programs on an unsolicited basis. This control, termed an unsolicited key-in, is initiated by the operator activating the INTERRUPT switch on the Sigma 2/3 Processor Control Panels. This action causes an interrupt into the Control Panel Task. The task sets a flag in the RBM Control Task status word, and then issues a Write Direct to trigger the RBM Control Task. The Control Panel 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. Each key-in must be terminated with the NEW LINE (NL) code. The backspace (¢) and delete (EOM) 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.

Specific key-in responses under RBM are:

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

where

oplb is an assigned operational label.

DFN is a decimal number (00 through 53).

P is an optional permanent change until system reboot.

BL oplb = oplb [,P] Alternate version of BL oplb = DFN[,P]

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

BT dn, track Add track number "track" to the Alternate Track Pool for device dn. If the Alternate Track Pool is not large enough or if dn is not a RAD device, an error message will be written.

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 7 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.

CP Clear card punch and simulate an unusual end condition in the punch. The key-in is required if the card punch fails to recover after a JAM A or JAM B. Operator should first manually clear the punch and restore it to READY, then interrupt and key in CP. The last (faulty) card will be repunched and cards in the normal stacker will be in the correct sequence.

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.

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.

[†]SYSGEN options (response to INC MISC query).

D[T]MM,DD[,YY][,HR,MN] Alternate version of D [T] MM/DD[/YY][,HRMN]

DR dn[t] 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'1CF' and sector X'200' will be dumped. This key-in applies only to the 7202 and 7204 RADs.

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

F Dump the contents of the File Control Table number (set in the DATA switches) on the operator's console. DATA switch value is DFN in hex and must be a SYSGEN number.

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 idle 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 assigned operational label.

DFN is a decimal number (00 through 53).

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 corrector cards from background device CC. (See Appendix G for the format of the corrector cards.) To patch program segments, DATA switch 0 must be placed in the "1" state. This causes RBM to type !!BEGIN SEG xx, where xx is the segment number (xx equals zero for the root), and go into an idle state after each segment is loaded. Correctors can then be loaded to the segment following an H key-in. An S key-in will cause RBM to resume operation. Correctors modifying foreground must be preceded by an FG key-in.

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

Lar,dn[,wp] Area mnemonic "ar", with a write protect code of "wp", will be written on sector 1 of device dn and sector 2 will be written with zeros. "wp" must be one of the following:

blank, D, or N	no write protect
B	background write only
F	foreground write only
R	RBM write only

The L (Label) key-in is an implicit 'M' key-in. Error conditions and causes are described under the !!KEY ERROR message in Table 6.

M ar,dn Mount area "ar" on device "dn". The operator must mount the disk pack containing area "ar" on device "dn" before making this key-in. Unless "ar" is "Xn" the disk pack will be read to determine if it actually is area "ar". If this is true, area "ar" will be added to the Master Dictionary and made available for general use, including use by the RAD Editor and M:ASSIGN. Error conditions are described in !!KEY ERROR message in Table 6. If an error occurs, area "ar" will be undefined and any areas implicitly "dismounted" will be undefined.

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.

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.)

[t] SYSGEN options (response to INC MISC query).

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 idle state.

T HRMN Reset the RBM system time.

T HR,MN Alternate version of T HRMN.

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 active, it will also be terminated.

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

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 7.) 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 7 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 B) using the absolute address given in Table 7. This is a useful technique for an absolute foreground program assembly, or for a processor or other programs that are self-relocating.

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.

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

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[†] and will then release the temp space and return to the caller by a call to the Monitor service routine M:POP. Any calls to the above Monitor service routines will require use of the RAD; therefore, the requesting task must provide in its TCB for use of the RAD. In addition, particular attention should be given to the maximum temporary stack requirements of these routines.

SERVICE ROUTINES

M:IOEX (General I/O Driver)

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

ADRLST is a pointer to the argument list, which is a set of two, three, or four consecutive words in the user's

[†] 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.

Table 7. Transfer Vector for Monitor Services

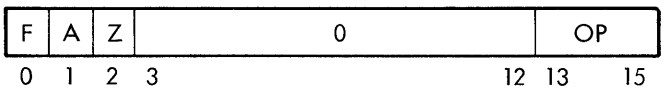
Address		ADRL for	Purpose of This Routine	Words of Temp Required	
Dec.	Hex.			Min.	Max.
199	C7	M:FSAVE	M:SAVE Function if Register Previously Saved	0	0
200	C8	M:IOEX	Device-Dependent I/O Driver	16	16
201	C9	M:READ	Device-Independent Read Routine	25	38
202	CA	M:WRITE	Device-Independent Write Routine	25	38
203	CB	M:CTRL†	Device-Independent Control Routine	16	43
204	CC	M:DATIME†	Calendar Date and Time of Day	18	18
205	CD	M:TERM	Normal Termination of Background	0	0
206	CE	M:ABORT	Abnormal Termination of Background	0	0
207	CF	M:SAVE	Save Registers on Real-Time Interrupt	0	0
208	D0	M:EXIT	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	Checkpoint/Restart Background	0	52
212	D4	M:LOAD†	Load Nonresident Foreground	13	13
213	D5	M:OPEN†	Open Blocking Buffer for RAD File	13	13
214	D6	M:CLOSE†	Close Blocking Buffer for RAD File	14	14
215	D7	M:DKEYS	Read Data Keys	0	0
216	D8	M:WAIT†	Execute Wait Loop From Background	15	51
217	D9	M:SEGLD	Load Overlay Segment	35	63
218	DA	M:DEFINE†	Define RAD Files in Background Temp Area	13	13
219	DB	M:ASSIGN†	Assign Operational Labels	18	56
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†	Reserve or Release Peripherals	20	51
224	E0	M:DOW†	Diagnostic Output Routine	13	51
225	E1	M:COC†	Communications Handler	25	25

†These routines are nonresident RBM overlays. All nonresident RBM overlays require a minimum of 13 temp cells 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. For example, M:READ (25) may call M:OPEN (13) to open a blocking buffer. This call would require 38 temp cells.
3. Under normal usage, M:SEGLD requires a maximum of 35 temp cells. However, 61 are required to output the message !!BEGIN SEG xx. This is an RBM assembly option (i.e., Debug = yes).
4. M:CKREST requires 52 temp cells if the checkpoint is performed at the priority level of the calling task.
5. Use of any device that has nonresident pre-I/O and post-I/O edit routines and nonresident error recovery routines associated with it requires 38 temp cells by M:READ/M:WRITE. These include KP, PT, LP, B7, CR, and CP.

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.

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

A = 0 if no AIO Receiver is specified (three-word call, then).

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

Z = 0 if acknowledged on channel end only.

OP is the code for the operation to be performed:

0 for SIO

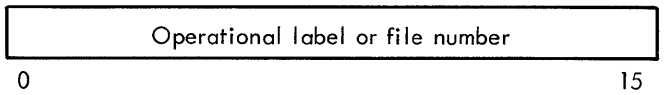
1 for TIO

2 for TDV

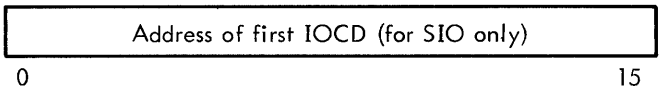
3 for HIO

4 for "check previous data transfer"

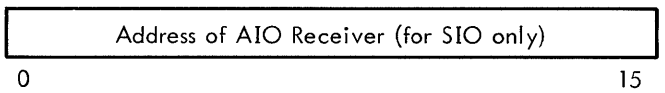
word 1



word 2



word 3



Return is to the location in the L register on the call to M:IOEX. B register 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 8. In this table, DSB stands for Device Status Byte, OSB for Operational Status Byte, Byte Count Residue is from the even I/O channel register at channel end, and Dev.No. stands for the device number of the current device.

Note that no I/O error recovery has been 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.

The user can use M:IOEX to read/write on the RAD or any peripheral device that uses standard Xerox Sigma 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/Write" (op code of 4) after each read or write request.

The following 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.

Table 8. 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 equals zero	0	0	0	——		2		0
SIO	SIO cannot be accepted[†]	0	1	0	Current file Number		TIO	Dev. No.	0
	Channel busy[†]	0	0	0	Active file Number		DSB	Dev. No.	-1
	Successful initiation	0	0	0	Current file Number		SIO DSB	Dev. No.	0
TIO	SIO cannot be accepted	0	1	0	Current file Number		TIO DSB	Dev. No.	——
	Other	0	0	0					
TDV	Device abnormal condition	0	1	0	Current file Number		TDV DSB	Dev. No.	——
	Device normal condition	0	0	0					
HIO	Device operating when HIO received	0	1	0	Current file Number		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 file Number		SIO DSB	Dev. No.	——
	I/O completed unusual end	0	1	0	E Flag (Bit 7)	OSB	AIO DSB	Dev. No.	Byte Count Residue
	I/O completed normal end	0	0	0					

[†]Use BXNC to test both conditions simultaneously.

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 returned.

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 allocated. 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).

If the request for I/O action is for an odd number of bytes, the order byte must be properly set in the right half of the word, as specified in the Sigma 2 and Sigma 3 Computer Reference Manuals. M:IOEX does not move any data or order bytes.

When using foreground data chaining, it is very important to set the interrupt flags on all IOCDs except for the one pointing to the "order" byte, 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.

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.
3. Background programs must specify an interrupt.

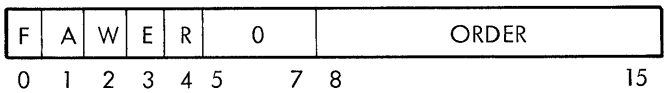
M:READ (General Read Routine)

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
```

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.

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

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

A = 0 if no AIO Receiver is specified.

W = 1 if wait for completion is unconditional.

W = 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.[t]

E = 0 if no error recovery is to be attempted.[t]

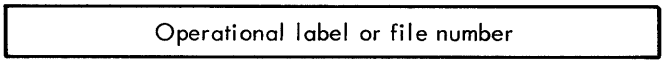
R = 1 if a RAD granule displacement is specified (can only be specified for random-access RAD files).

R = 0 if a RAD granule displacement is not specified.

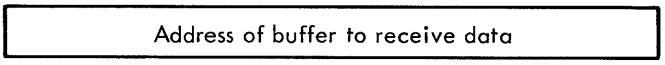
ORDER is one of the following permissible pseudo input orders:

Order	Operation
X'00'	Return information about this device and file.
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).

word 1



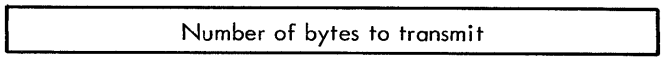
word 2



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

[t] For magnetic tapes, RAD, or disk pack, five attempts for error recovery will be made if E is specified. If I/O without a WAIT is specified, error recovery will not be performed until a "Check/Write" is issued by the user.

word 3



Byte count must be an even number when reading from RAD files and cannot exceed 65,536. 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 granule displacement (see word 5).

word 5

If an AIO address is specified (A = 1 in word 0), word 5 indicates the displacement of the granule from the start of the file (starting with a displacement of zero) where the I/O transfer begins. Word 4 is the RAD granule displacement if A = 0.

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 9. I/O completion codes are listed in Table 10. 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 (unless M = 1 in word 0). 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.

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

Table 9. 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	t
	Calling sequence error	Return immediately	-1	4	t
	Operational label is set to zero	Return immediately	0	2	t
	Unrecoverable I/O error	Return after error recovery attempt, if any	-1	1	t
	Illegal sequence of RAD operations	Return immediately	0	9	t
	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. 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 no device recognized)	Return immediately	-1	-1	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	Byte count
Initiate and wait	Channel and device are free and automatic	Initiate I/O and wait for completion			
	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

Table 10. 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	Unrecoverable I/O.	If error recovery was specified, the maximum number of retries have been unsuccessfully attempted.
0	2	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	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	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	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	Illegal sequence of RAD operations.	Significant only for sequential RAD files.
0	10	Blocking buffer unavailable.	Significant only for blocked or compressed sequential RAD files.

code is X'00', the following device status information is returned:

Register	Status Information
A	Device name (EBCDIC).
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.

M:READ FUNCTIONS

M:READ is designed to read one physical 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.

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.

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 Binary From Paper Tape. The Read Binary order for paper tape is "read ignoring leader". 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-of-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 NEW LINE 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 EBCDIC or binary. Binary and EBCDIC 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.

"End-of-file" status is returned when a file mark is sensed on the magnetic tape; "end-of-tape" status, when the physical end-of-tape mark is sensed and standard error recovery is specified. If both are sensed at the same time, the "end-of-tape" status is returned.

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, binary and EBCDIC 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.

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. Binary and EBCDIC modes are again identical. 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.

If the Read begins or extends beyond the file's ending boundary, no data is transmitted and "end-of-tape" status is returned. This is true whether error recovery is specified or not. The granule displacement must always be specified; if not, a "calling sequence error" is returned.

Note: For all RAD files, no transfer will be initiated that crosses 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.

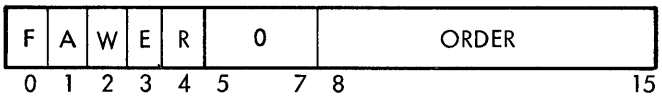
M:WRITE (General Write Routine)

M:WRITE provides 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
```

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.

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

A = 1 if an AIO Receiver address is specified.

A = 0 if no AIO Receiver address is specified.

Note: only a foreground operation can specify this.

W = 1 if wait for completion is unconditional.

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

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

E = 1 if standard error recovery is to be performed at channel end for this operation.[†]

E = 0 if no error recovery is to be attempted.

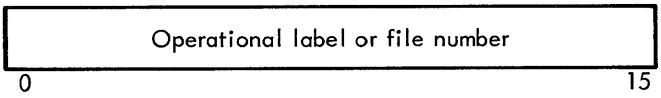
R = 1 if a RAD granule displacement is specified (can only be specified for random-access RAD files).[†]

R = 0 if a RAD granule displacement is not specified.

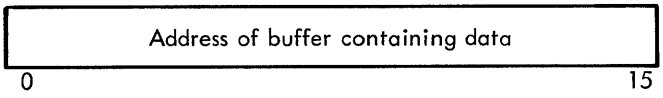
ORDER is one of the following pseudo order bytes:

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

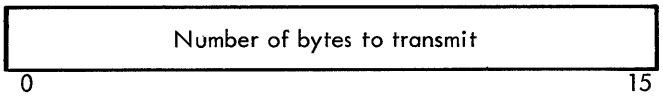
word 1



word 2



word 3

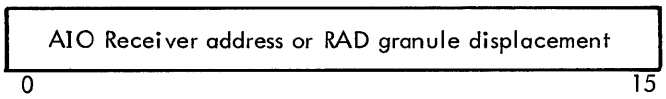


The byte count must be an even number when writing on RAD files and may not exceed 65,536. It may be either even or odd for all other devices, but cannot exceed 8192 bytes. If an odd byte count is requested, the first byte is written from the right half of the word and the left half is ignored. If an 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.

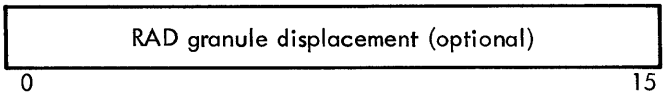
[†]For magnetic tapes, RAD, or disk pack, five attempts for error recovery will be made if E is specified. If I/O without a WAIT is specified, error recovery will not be performed until a "Check/Write" is issued by the user.

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 (see word 5).

word 5



If an AIO address is specified (A = 1, word 0), word 5 indicates the displacement of the granule from the start of the file (starting with a displacement of zero) where the I/O transfer begins. If A = 0, word 4 is the RAD granule displacement. See Chapter 6 for further details.

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.

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
RAD (sequential file)	Logical file mark
RAD (random file)	Logical record 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 sequential 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.) The Write End-of-File order for any RAD file causes an implicit call to M:CLOSE and any data written in the blocking buffer will be output on the RAD.

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 inserted as the last byte (if NEW LINE is not already present). If there are more than 85 printable characters, those beyond 85 are ignored, and a status of "incorrect length" is returned.

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 Binary or 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 XDS printers.

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 specified (up to the allowable maximum) is always written.† 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

†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 Systems Reference Manual, Publication 90 09 78).

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 Random-Access RAD Files. Although a granule size may be specified when a random file is defined, the size does not restrict 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. The sector displacement must always be specified; if not, a "calling sequence error" status is returned.

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.

Note: For all RAD files, no transfers 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.

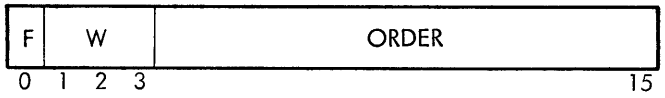
M:CTRL (General Control Routine)

M:CTRL provides device-independent positioning capabilities for magnetic tapes (both 7-track and 9-track) and for sequential 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
```

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

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

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

W = 1 if wait for operation is to be initiated.[†]

W = 0 if no wait for operation is to be initiated when device/channel is busy.[†]

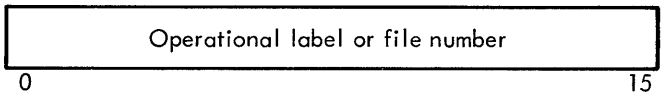
ORDER is one of the following pseudo order bytes:

Order	Operation
X'EB'	Space Record Backward
X'EF'	Space Record Forward
X'FB'	Space File Backward

[†]The W flag has a different function for M:CTRL than for M:READ/M:WRITE. If the operation is initiated, control will not be restored to the calling task until the operation is complete.

Order	Operation
X'FF'	Space File Forward
X'2B'	Rewind Off Line
X'3B'	Rewind On Line

word 1



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.

Note: For random-access 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 sequential 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 either an end-of-file or end-of-tape marker, EOF or EOT 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 sequential 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 to the start of the next physical record. If the record skipped was either an end-of-file or end-of-tape marker, EOF or EOT 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 sequential 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 sequential 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 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 either the start of the next file or the end-of-tape mark, whichever is encountered first. Either a status of EOF or EOT is returned.

For sequential 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 sequential 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 sequential 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.

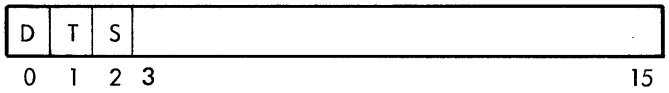
M:DATIME (Calendar Date and Time of Day)

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
```

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.

D = 0 if calendar date is not required.

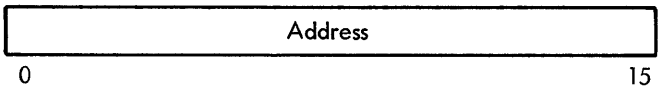
T = 1 if return time of day is specified.

T = 0 if time of day is not required.

S = 1 if date and time are supplied by the user (in Address and Address + 1).

S = 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 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:

Date:	M	M
		D
	D	
	Y	Y
	Δ	Δ
Time:	H	R
	M	N

2 blanks are supplied when both date and time are requested

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.

M:TERM (Normal Exit from Background Programs)

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

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

M:TERM FUNCTIONS

If called by a foreground program, control will be transferred to M:EXIT to perform the exit sequence for that task. 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 (Background 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
```

CODE is a word of EBCDIC 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, M:EXIT is called to perform the exit functions. If the calling task occupies the nonresident foreground area, it will be disabled and 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 Clock 1 has been reserved for RBM accounting, M:SAVE will record the start time of the first interrupting foreground task and will trigger the RBM Control Task to calculate foreground run time.

An additional entry point, M:FSAVE, is available for users of the Sigma 3 optional instruction, Store Multiple. This entry point, with an address literal in cell X'C7', assumes that all registers 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
DATA    -1
DATA    RETURN
```

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.

If the two optional data words (DATA - 1 and DATA RETURN) are used, M:EXIT restores all registers and context, except overflow and carry and the interrupt status; but instead of performing the hardware exit, M:EXIT branches to RETURN.

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: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.

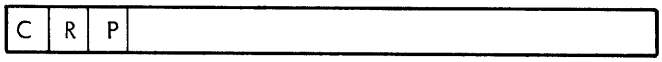
M:CKREST (Checkpoint/Restart Background)

M:CKREST checkpoints the background (i.e., writes it out onto 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
```

ADRLST is a pointer to an argument list, as follows:

word 0



where

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

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

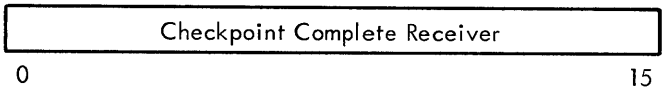
R = 1 if a Checkpoint Complete Receiver is to be informed when the checkpoint is complete. (Valid only if C = 1 and P = 0.)

R = 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).

P = 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"[t] 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.

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.

---

[t]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.

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)

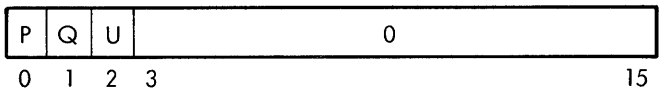
M:LOAD initiates the loading of the root segment of a resident or nonresident foreground program by entering the requested program name into the queue stack. It also initiates the loading of the root segment of a resident or nonresident foreground program or background processor upon request from the Job Control Processor. It releases (unloads) 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
```

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.)

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

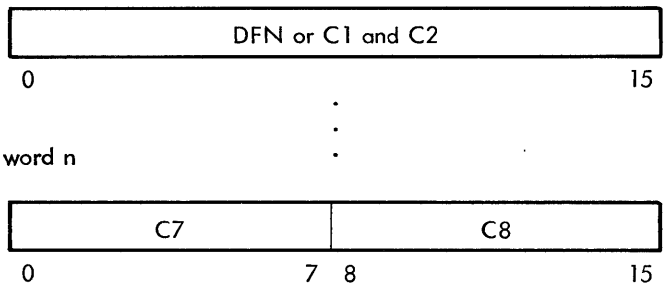
Q = 1 indicates the request is to be queued if it cannot be satisfied now.

Q = 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.

U = 0 indicates a load operation.

word 1



where

DFN is the device-file number.

C1-C8 is the program name (must be 8 characters, including trailing blanks).

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.

It is essential that each nonresident program executed in the nonresident foreground area terminate itself by a call to M:LOAD 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.

For an unload request, M:LOAD 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)

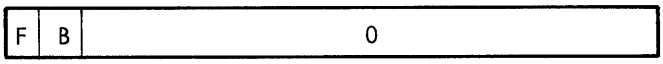
M:OPEN reserves a blocking buffer from a buffer pool or a specified location, for a sequential blocked RAD file to which an operational label or device unit number had previously been assigned.

The calling sequence is

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

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).

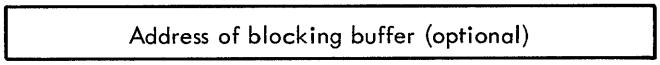
F = 0 if an operational label or device unit number is specified.

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

B = 0 if no blocking buffer location is included, in which case M:OPEN attempts to find space in the task's buffer pool.

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 File Control Table. If no open request has been performed for a sequential blocked 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.

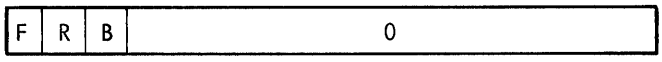
M:CLOSE (RAD File Release)

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
RCPYI   P,L
B       M:CLOSE
```

ADRLST is a pointer to the argument list, as follows:

word 0



where

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

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

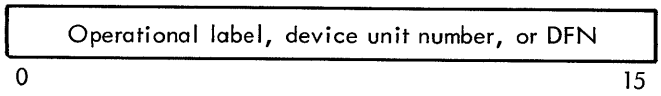
R = 1 if the device-file number is to be released.

R = 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).

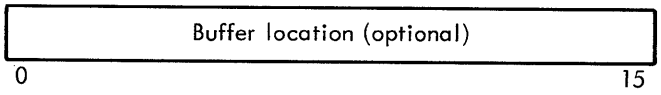
B = 1 is a buffer is specified.

B = 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 the following completion status.

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 permenent assignment, the label is set "unassigned". 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 (sequential file only it must also be written onto the RAD. To accomplish the writing, M:CLOSE requires a buffer, one sector in length, into which the file dictionary 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.

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)

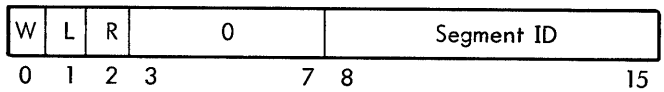
M:SEGLD loads and/or executes an overlay segment, for either the foreground or background, from a file previously prepared and saved on the RAD by the Overlay Loader or the Absolute Loader.

The calling sequence is

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

ADRLST is a pointer to the argument list.

word 0



where

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

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

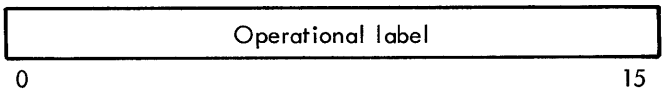
L = 1 control is to be transferred to the transfer address of the segment just loaded (valid only if W = 1).

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

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

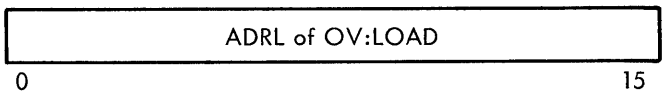
R = 0 no "loading complete" receiver.

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.

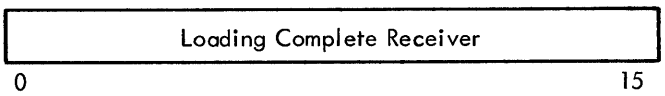
word 2



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.
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)

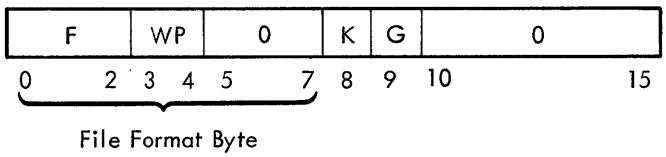
M:DEFINE allocates a portion of the background temporary file area on the RAD for temporary use by the designated operational label or device unit number. This call is applicable to foreground operations only if the file is previously assigned to a permanent RAD file. The calling sequence is

```
LDA     PTR     (FORTRAN programs only)
LDX     ADRLST
RCPYI   P,L
B       M:DEFINE
```

PTR is the absolute address of the FORTRAN Associated Variable. 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
110	random

WP = 11 if RBM write protection is specified.

WP = 10 if foreground write protection is specified.

WP = 01 if background write protection is specified.

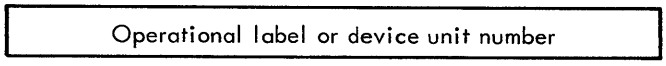
WP = 00 if write protection is not desired.

K = 1 if the A register contains the address of the FORTRAN Associated Variable.

K = 0 if FORTRAN Associated Variable 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).

word 1

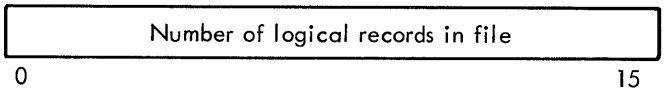


where

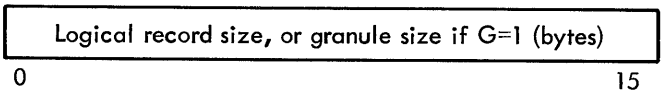
operational labels are EBCDIC

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, 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.
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	If K = 1, attempted to define previously defined file using inconsistent parameters.

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 RAD Files)

M:ASSIGN performs equivalence between an operational label or FORTRAN device unit number, and

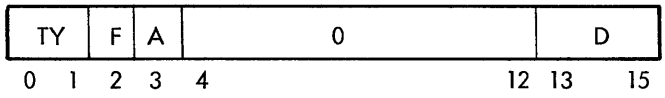
1. A RAD area.
2. A file name within a RAD area.
3. A device-file number.
4. Another operational label or device unit number.

The calling sequence is

```
LDX     ADRLST
RCPYI   P,L
B       M:ASSIGN
```

ADRLST is a pointer to an argument list of two to eight words, as follows:

word 0



where

TY = 00 if the label is to be assigned to another label.

TY = 01 if the label is to be assigned to a device-file number.

TY = 10 if the label is to be assigned to a RAD area.

TY = 11 if the label is to be assigned to a file within a RAD area.

F = 0 if the label is a background operational label.

F = 1 if the label is a foreground operational label.

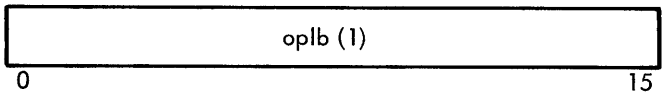
A = 1 if the two-letter area mnemonic is contained in word 3; otherwise, D will specify the area. If A is set, D will be ignored. A must always be set for areas other than SP, SD, SL, UP, UD, UL, BT, and CP.

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

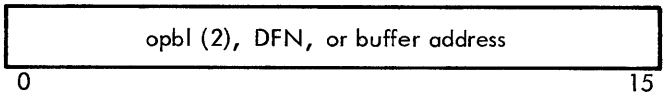
No named files may exist in either the Checkpoint or Background Temp areas. D is ignored for TY = 00 or 01.

word 1



where oplb (1) is the operational label or device unit to be assigned.

word 2



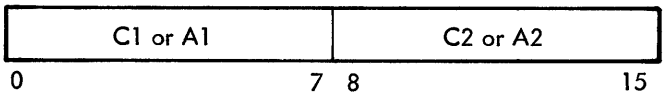
where

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.

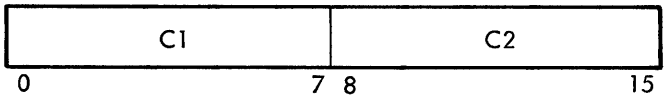
buffer address is the first word address of a buffer (equal to one blocking buffer in length) that will be used by M:ASSIGN as temporary storage for the appropriate RAD area dictionary. This is meaningful only for TY = 11.

word 3

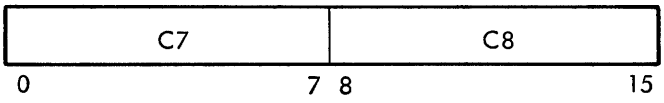


If A (of first word of argument list) = 1, word 3 contains the two-letter area mnemonic, A1 and A2; otherwise, word 3 contains the first two characters of the file name, as continued below:

word 3 + A



word 6 + 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).
2	Invalid oplb (2) 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 standard record size of this device.

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).

TY = 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.

TY = 10 or 11 oplb (1) is assigned to a currently unused RAD DFN, which in turn is linked via the RAD dictionaries to a file on the RAD. This RAD file may be either an entire RAD area (e.g., system processor) for TY = 10, or an individual file within an area (e.g., XSYMBOL) for TY = 11. The RAD area must have been allocated at SYSGEN (error return A = 5). The buffer address (TY = 11 only) 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, sequential 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 Without Transfer)

M:RES allocates storage in a temporary stack, saves the previous value of B, and sets B to the first word address of temporary area being allocated. The calling sequence for dynamic allocation of storage is

```
RCPYI    P,T
B        *$+3
DATA     n
DATA     0
ADRL     M:RES
```

where n is the number of cells to be reserved.

T must point to the background if it is a background program.

A TS abort will occur if more temporary storage is requested than is available.

The calling sequence for nondynamic allocation of storage is

```
RCPYI    P,T
B        *$+3
DATA     n
ADRL     TEMP
ADRL     M:RES
```

where TEMP is the address of n reserved locations at the end of the calling program. This area must not contain any code or literals.

Upon return, the B register contains the pointer to the new temporary storage stack. Locations 0 and 1 relative to the base register are used by the storage allocation routines and may not be used by other routines. Location 2 relative to the base (the return address for M:POP) is set to M:ABORT.

The calling program can set up its own exit through M:POP via the following.

```
LDA      =RETURN
STA      2,,1
```

The L and X registers are unaffected.

M:POP (Temporary Storage Release Routine)

A call to M:POP is made to release the current TEMP 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
B       M:POP
```

where RETURN is the location to which return will be made after the stack is released.

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: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 location 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

E = positive if label is found

If E is positive, the following information is provided.

Register	Contents
X	Device-file number
E	IOCT entry address[t]
A	Operational label table entry[t]

Note: This routine is used primarily by the RBM and certain processors. It will seldom be needed by user programs.

[t] See the chapter on SYSGEN for a discussion of the I/O Control Table and the Operational Label Table.

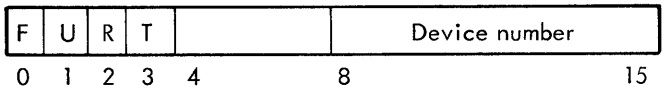
M:RSVP (Reserve or Release Peripherals)

M:RSVP reserves a peripheral device for foreground use only, until the foreground voluntarily releases the device.

```
LDX     ADRLST
RCPYI   P,L
B       M:RSVP
```

ADRLST is the pointer to the argument list, which consists of three 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".

F = 0 if request is "release to background".

U = 1 if request is for an unconditional reserve, where operator intervention is not required.

U = 0 if request is for a conditional reserve, where operator intervention is required.

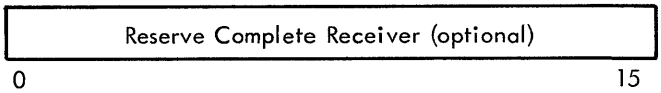
R = 1 if a receiver is to be entered when the conditional reserve is completed (only meaningful if U = 0).

R = 0 if no such receiver is to be used.

T = 0 if a device type is not specified.

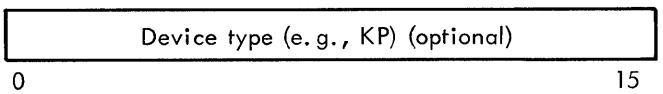
T = 1 if a device type is specified (used to distinguish KP40 from PT40).

word 1



A Reserve Complete Receiver should be used like an 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. If R = 0, word 1 contains the device type (see word 2).

word 2



Return 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.

A = 1 if the request is acknowledged but operator intervention is required. If a Reserve Complete Receiver is specified, it 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).

A = 2 if the device is not associated with a background file.

A = -1 if the request cannot be honored because a prior request to reserve this device has been made, if the request is to release an unreserved device, or if the reserve peripheral table (RSVTBL) is full. (See "Limitations" below.)

M:RSVP 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., !!FG RESERVE, LP02).

If the request is for a conditional reserve, a message is output to inform the operator of the request (e.g., !!FG REQUEST, 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. A message is now output to acknowledge the reserve action (e.g., !!FG RESERVE, CR03). If the Reserve Complete Receiver is specified, it will be entered at this point.

Release. The peripheral device can be released for background use by a call to M:RSVP to release the device. The peripheral device specified will now be available for background use. A message will be output to inform the operator of the release action (e.g., !!BK RELEASE,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 five requests at a time, which is felt to be a realistic limitation.

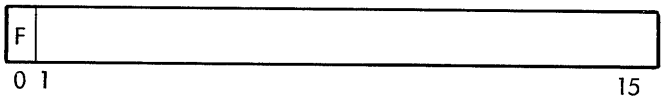
M:DOW (Diagnostic Output Writer)

Currently, multitask use of the same file may result in a conflict situation whereby a task is unable to output a message because a lower priority task has control of the file. M:DOW allows the use of an active file for the purpose of outputting alarms. The calling sequence is

```
LDX     ADRLST
RCPYI   P,L
B       M:DOW
```

ADRLST is a pointer to the four-word argument list as shown below:

word 0

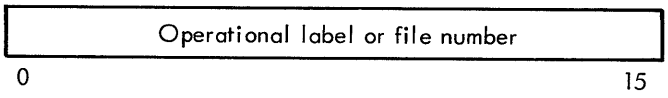


where

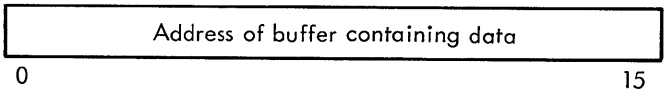
F = 1 if a device file number is specified.

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

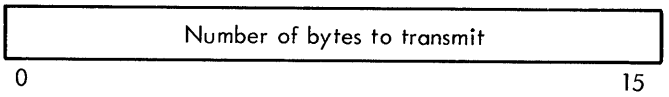
word 1



word 2



word 3



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. The method of returning and the status returned are the same as described under M:READ/M:WRITE.

M:DOW FUNCTIONS

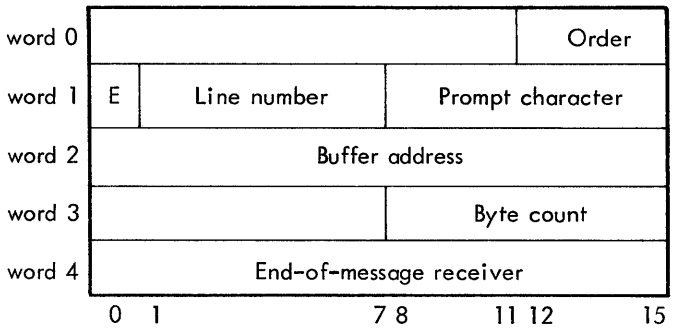
If the file to be used is currently active, M:DOW will wait until end-action-pending and will then clear the active file and the end-action-pending flags. The call will be translated to an equivalent call to M:WRITE which will output the alarm. The buffer data are assumed to be EBCDIC.

M:COC (Character-Oriented Communications)

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

ADRLST is a pointer to the argument list, as follows:



where

Order (bits 12-15) is as follows:

Order	Operation
0	Check status of line
1	Write n[t] bytes, no editing
2	Read n[t] bytes, no editing
3	Send break character (long-space)
4	Check previous read or write
5	Write message of up to n[t] bytes, edited
6	Read message of up to n[t] bytes, edited
7	Disconnect line (turn off data set)
8	Connect line

E is 1 if an end-of-message (EOM) receiver is specified; is 0 if no EOM receiver is specified.

Prompt character is meaningful 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 editing or translation will be performed (overrides setting of bits 9 and 10).

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 5 for a more detailed discussion of AIO receivers.)

Return 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 (which must be 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). Only the line number is needed in the argument list.

[t] $0 < n \leq 255$.

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	Return	0	Completion code	Byte count
Connect or disconnect	Successful connection	Connect and return	0	0	Line no.
Check status	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 (O and C bits)
14-15	Transmitter status (O and C bits)

Table 14. Line Mode

A Register Value	Meaning
0	Line is disconnected
1	Output mode
2	Output prompt character and then switch to input
3	Input mode
4	Inactive mode
5	Message complete

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.

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 data set is disconnected, but the send and receive modules remain connected. The logical line mode is cleared (i.e., disconnected).

8 Connect line. The logical line mode is set to "inactive" and the options are initialized. The connect 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 to determine when a line has connected.

M:COC FUNCTION

Once the RCOC initialization routine has prepared the communication equipment, the status of each line is "disconnected". All input and output are rejected until the line is connected. If the line is dedicated, only a "connect line" call to M:COC is required. If the line must be dialed-in (using M:IOEX), the dial operation must precede the "connect line" call to M:COC. The connect sets the line status to "inactive" (i.e., available for I/O transfers). I/O operations are initialized sequentially, and when completed, the line status is set to "message complete". At this point the line is still busy and can be cleared (i.e., set to "inactive") only by a call to M:COC to check the status of the previous operation (order 4). The call "check operation" is not required after a check status, a connect or a disconnect operation. A disconnect operation sets the line status to "disconnected", and the line must be reconnected before it can be used again (see Appendix F).

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

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 recovery (optional).
- Software checking of background and Monitor.
- 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 sequential 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
M9	10
PT	820
BR	3
BP	3
M7	10
RD 7202, 7204	3
RD 7242	4
PL	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            (device status byte from AIO in bits 0-7;
                         device number in bits 8-15)
B       AIO Receiver address
```

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 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.

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.

SEQUENTIAL FILES

1. Sequential RAD files are available to foreground and background tasks.

2. Sequential RAD files are available to routines M:READ, M:WRITE, and M:CTRL, but not to M:IOEX.

3. Sequential RAD files can be blocked (with more than one logical record per sector) if the logical record size is less than or equal to half the RAD sector size. The Monitor I/O routines do the blocking and unblocking.

4. 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.

5. 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.

6. 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).

7. 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.

8. Sequential RAD files (except compressed files) can be spaced forward or backward by logical records.

9. Sequential RAD files can be positioned by !REWIND, !FBACK, and !FSKIP commands.

10. Sequential RAD files can request 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 indicating whether the AIO Receiver is to be acknowledged or not, (see M:READ/M:WRITE status returns).

11. RAD transfers must consist of an even number of bytes.

12. 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 (for standard assignments) or through Monitor service calls at execution time for nonstandard assignments.

13. When the operational label is defined or assigned to a permanent file, it is automatically positioned at the BOT.

14. As on magnetic tape, the only record that can be written at the EOT is the logical file mark.

RANDOM FILES

1. Random files are available to foreground and background jobs.

2. Random files are available to routines M:READ and M:WRITE, but not to M:CTRL or M:IOEX.

3. All 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).

4. All positioning commands such as !REWIND, !FSKIP, !WRITE EOF, etc., are meaningless.

5. The transfer of any number of bytes (up to a maximum of 65, 536) may be requested, provided that the byte count is an even number and the transfer will not extend past the file boundary.

6. Operational labels can be equated to permanent files on the RAD or can be allocated from available temporary RAD space. This can be accomplished either through control commands (for standard assignments) or through Monitor service calls at execution time for nonstandard assignments.

7. 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.

8. Random files cannot be blocked or compressed, unless the user program performs its own blocking/deblocking or compression/decompression.

9. BOT is defined as the first sector of the file. EOT is defined as the last sector +1 of the file. EOF has no meaning in random files except for mapping purposes.

10. Requests for a foreground AIO Receiver at channel end will always be acknowledged.

GRANULES

Granules are the minimum physical amount of data that are transferred in a read or write operation from or to random

RAD/disk pack files. 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.

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 15, plus: 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. 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.

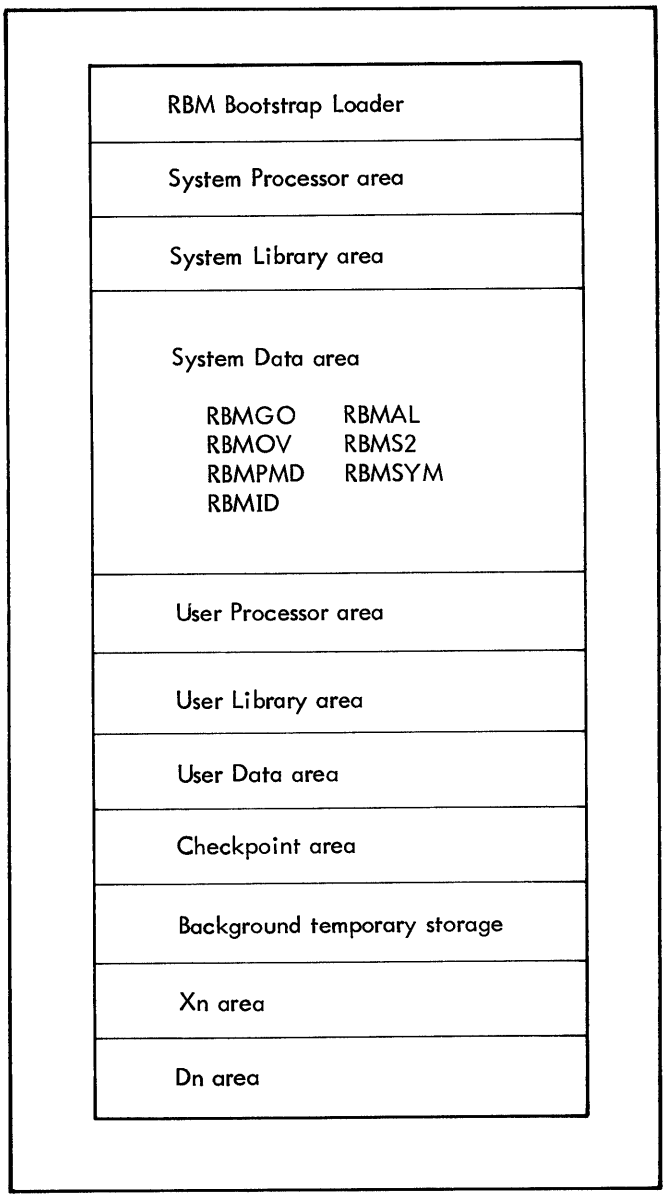


Figure 4. RAD Allocation

6. REAL-TIME PROGRAMMING

FOREGROUND PROGRAMS

Under the Sigma 2/3 RBM, a foreground program is one that operates in protected memory, utilizes foreground operational labels or device unit numbers, and has access to privileged Sigma 2/3 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 it important that this distinction be understood.

RESIDENT FOREGROUND PROGRAM

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 PROGRAM

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 six cells 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

Memory Parity Error Task

Protection Violation Task

Multiply Exception Task (optional)

Divide Exception Task (optional)

Input/Output Task

Control Panel Task

Counters = 0 (optional)

Real-Time Task(s), if any lower than I/O

RBM Control Task (lowest hardware level)

Background (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.
- Loads and links and branches to power-on overlay.

- Disarms all external and internal interrupt levels, then arms and enables all interrupt levels.

- Interrogates foreground mailbox X'C4' for a power-on receiver, and if one is specified, links and branches to it. An override task is one that services an interrupt generated at the override group level (dedicated interrupt locations X'100' to X'105'). The receiver for such a task may be specified by loading a resident task into foreground. This task must have a small initialization routine that sets the corresponding foreground mailbox to point to the real-time portion of the receiver. The various receiver mailbox addresses and their corresponding functions are as follows:

Address	Receiver Function
X'C3'	Power Off
X'C4'	Power On
X'C5'	Integral IOP Timeout
X'C6'	Watchdog Timeout

- Scans the Channel Status Table and, for any active I/O channel, sets Unusual End and Memory Parity Error flags and simulates an I/O interrupt.

- Retriggers any task that has its TCB address in the TCB chain.

- Restores protection registers.

- Triggers RBM to write the message !!POWER ON.

- Reloads the overlay region.

- Switches the dedicated interrupt location for any task that requires retriggering, so that the interrupt branches to a separate Power On Task. This separate Power On Task then branches to the point at which the task (at this interrupt level) was interrupted and subsequently switches the dedicated interrupt location back to its proper value.

- Exits the power failure task (i.e., the Power Off Task).

POWER OFF TASK

The Power Off Task performs the following operations:

- Saves the internal interrupt status.

- 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.

- Saves the RAD address of the RAD that has the system processor (SP) area.

- Interrogates foreground mailbox X'C3' 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

This task is responsible for examining memory parity errors and watchdog timeouts. If a memory parity error occurs while the background is active, the background program is aborted and the real-time foreground is not disturbed. The Machine Fault Task calls the reentrant Monitor routine M:ABORT which sets the flag for the S:ABORT subtask and triggers the RBM Control Task. S:ABORT then aborts the background and prints an error message.

If a memory parity error occurs while a foreground task is active and if the number of foreground parity errors has not exceeded the specified limit, the Machine Fault Task sets a flag to cause the RBM Control Task to output the following diagnostic:

```
!!FG PARITY ERR, TCB=FFFF, LOC=FFFF, A=FFFF,
X=FFFF, B=FFFF
```

Processing continues from the point where the parity error occurred.

If a memory parity error occurs while a foreground task is active and if the number of foreground parity errors has exceeded the specified limit, the Machine Fault Task does the following:

1. Disables the active task.
2. Sets a flag to cause the RBM Control Task to output a diagnostic.
3. Resets the foreground parity error counters.
4. Exits and simultaneously forces the active task to terminate.

The RBM Control Task outputs the following diagnostic:

```
!!FG PARITY ERX, TCB=FFFF, LOC=FFFF, A=FFFF,
X=FFFF, B=FFFF
```

where ERX indicates that the task has been disabled and terminated.

All tasks that do not use M:SAVE must set K:TCB correctly to guarantee proper recovery from a memory parity error.

If an integral IOP watchdog timeout occurs, the Machine Fault Task interrogates foreground mailbox X'C5' for an integral IOP timeout receiver. If a receiver is specified, the Machine Fault Task links and branches to it; otherwise, the Machine Fault Task enters the "wait" state. The

overflow bit will be set, and the control panel interrupt will be ineffective in clearing this "wait". When this happens, a Customer Engineer should be notified immediately.

If an external IOP watchdog timeout occurs (usually as the result of attempting direct I/O to an unrecognized device), the Machine Fault Task interrogates foreground mailbox X'C6' for a receiver. If a receiver is specified, the Machine Fault Task branches to it; otherwise the Machine Fault Task outputs the message

```
!!MACH. FAULT; TCB=FFFF, LOC=FFFF, A=FFFF,
X=FFFF, B=FFFF
```

It then changes the program status to set the overflow and carry when it exits, and attempts to continue with the foreground task. If this interrupt occurs twice for the same task, the Machine Fault Task triggers RBM to write the following message:

```
!!MACH. FAULX; TCB=FFFF, LOC=FFFF, A=FFFF,
X=FFFF, B=FFFF
```

It then disables and terminates the current foreground tasks.

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 Memory Parity Task.

Unavailable core is set "protected". 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

These tasks simulate and subsequently execute a Multiply or Divide instruction for Sigma 2/3 computers not equipped with Multiply/Divide hardware. They are not reentrant, so all lower interrupts are 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 exit 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 interrupts.

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.

Certain RAD I/O operations are subject to a minimum-seek algorithm. Under this algorithm, RAD seeks are not initiated until the RAD is positioned within two sectors of the first sector to be read. This prevents low-priority tasks from denying RAD access to high-priority tasks. The algorithm applies to all "wait" requests (see description of M:READ and M:WRITE in Chapter 4).

CONTROL PANEL TASK

A Control Panel Interrupt causes the Control Panel Task to set a flag for the RBM Control Task, trigger the task, and then exit from the Control Panel Task (about 40 to 50 microseconds of CPU time). The operator response is processed 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 different 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 operations 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 Sigma 2/3 exit procedure or may have these functions performed by the M:EXIT service routine.

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 task 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

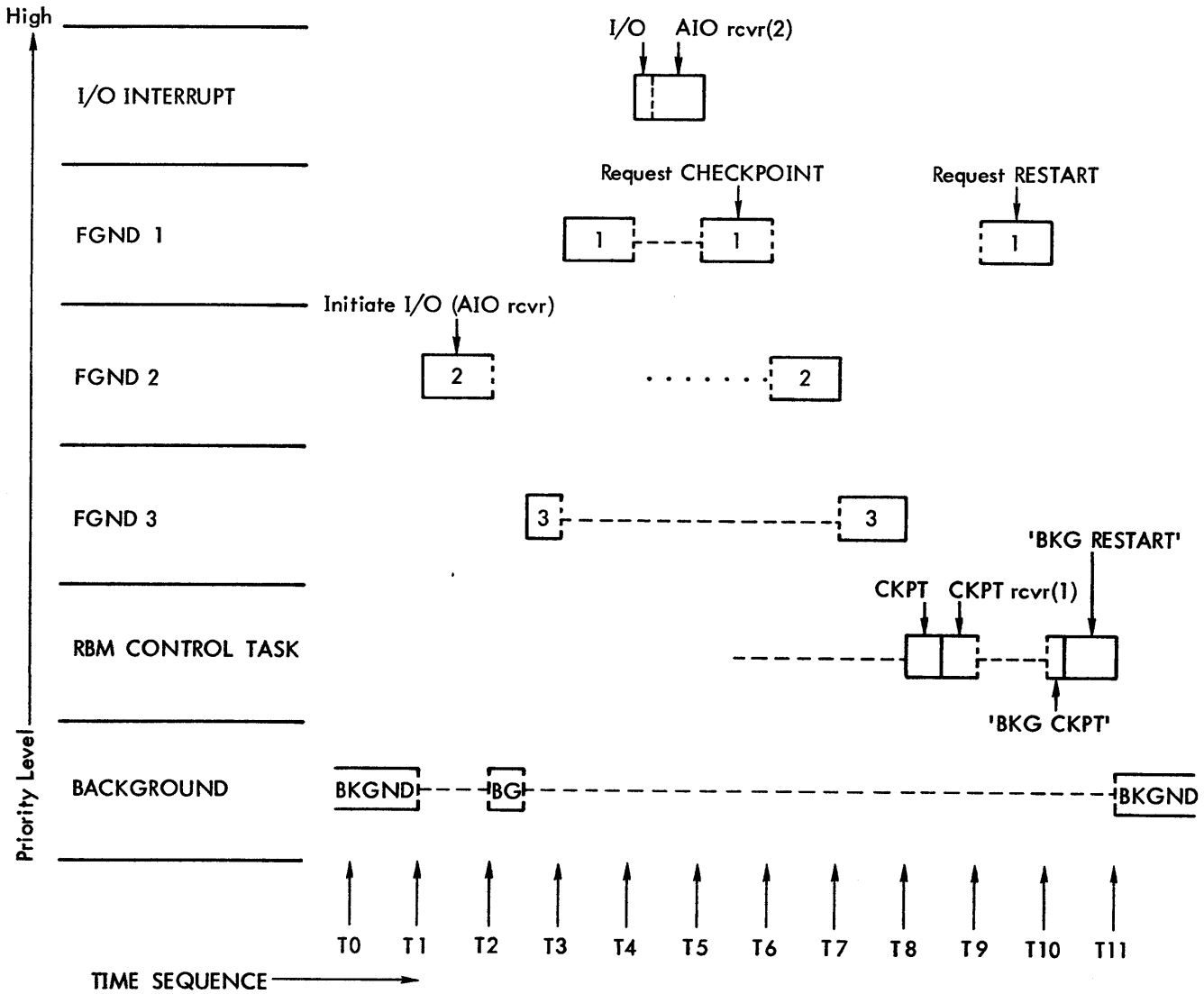
Foreground programs may be loaded into core for execution in any of several ways. All programs must reside on the RAD to be read into core memory for execution. They must be written onto the RAD by the Overlay Loader or the Absolute Loader.† In each of the methods described below, only the root is loaded into memory 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 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.

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 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 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 to be loaded into this area. Therefore, whenever a nonresident foreground program is loaded, all queue stack loading is suspended until the program occupying the nonresident foreground area releases the area by unloading.

†See the !ABS control command description in Chapter 2 for restrictions regarding the use of the Absolute Loader.



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 task'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.)

Two other methods of loading foreground programs are available. They involve control commands normally used by the background, are part of a background job stack, and 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 on either the System Processor or User Processor areas of the RAD. The user is responsible for avoiding the duplication of program names.

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 RESIDENT FOREGROUND PROGRAMS

Loading of real-time programs into their predefined RAD files can be accomplished by the Absolute Loader from the background job stack, or resident foreground programs can be written into their predefined RAD files by the Overlay Loader. It is not necessary to create the foreground programs when the system is created. However, to get the foreground program in absolute form will require either the use of the Overlay Loader or that the job be assembled in absolute as a self-contained package.

LOADING NONRESIDENT FOREGROUND PROGRAMS

Nonresident foreground programs are 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. Ordinarily, a program

releases space by a call to M:LOAD to "unload". However, a FORTRAN program has no means of performing this unload except by calling a special library routine. 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.

FOREGROUND INITIALIZATION

When a foreground program is loaded, it may either be initialized† by RBM or may have its own initialization routine (coded in assembly language). If the header of the foreground program contains a transfer address, RBM honors this address as the entry point to an initialization routine. This routine may arm and enable (or whatever) one or a number of related real-time interrupts. It can also set RAD files for subsequent use and set up initial values in core data tables. The initialization routine runs at the priority level of the RBM Control Task with the privileges of a foreground program. The initialization routine should make no calls on routines requiring temporary storage, since the RBM temp stack is the one in use. When foreground initialization is completed, the routine returns to RBM via a register copy of L to P. Foreground initialization routines will also be executed any time the system is rebooted from the RAD.

†See Overlay Loader options in Chapter 7.

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 17.

Table 17. 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: 0; 6: X; 7 - 15: Dedicated Interrupt Location	Assembler/Loader
2	0 - 3: 0001; 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)
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

Table 17. Task Control Block (TCB) (cont.)

Location	Contents	Set by
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	Number of buffers ($1 \leq n \leq 16$) (0 if unused).	Assembler/Loader
16	"Use" bits for buffers in pool (0 if unused).	M:OPEN or M:CLOSE
PSD + 0	Interrupt task status flags	Interruptsequence
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 continguous with the TCB.)	Assembler/Loader

where

ADRL PSD is 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 0007; 0 means yes, 1 means no. (T must be 0 if any Monitor service routines are used.)

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.

CODE is the interrupt system control code (as defined in the Sigma 2 and Sigma 3 Computer Reference Manuals), that indicates current or desired initial interrupt control status.

Buffer pool is an amount of space from one 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.

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 Sigma 2 and Sigma 3 Computer Reference Manuals under "Interrupt System Control."

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 routine 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 an 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. The transfer address given in the object deck 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.

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:

```
!!ABORT CODE XE, 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 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 with a priority level lower than the I/O priority level and operated without interrupts inhibited may use the Monitor I/O routines without any special restrictions. However, foreground tasks that have interrupts or have an interrupt level higher than the I/O priority level 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

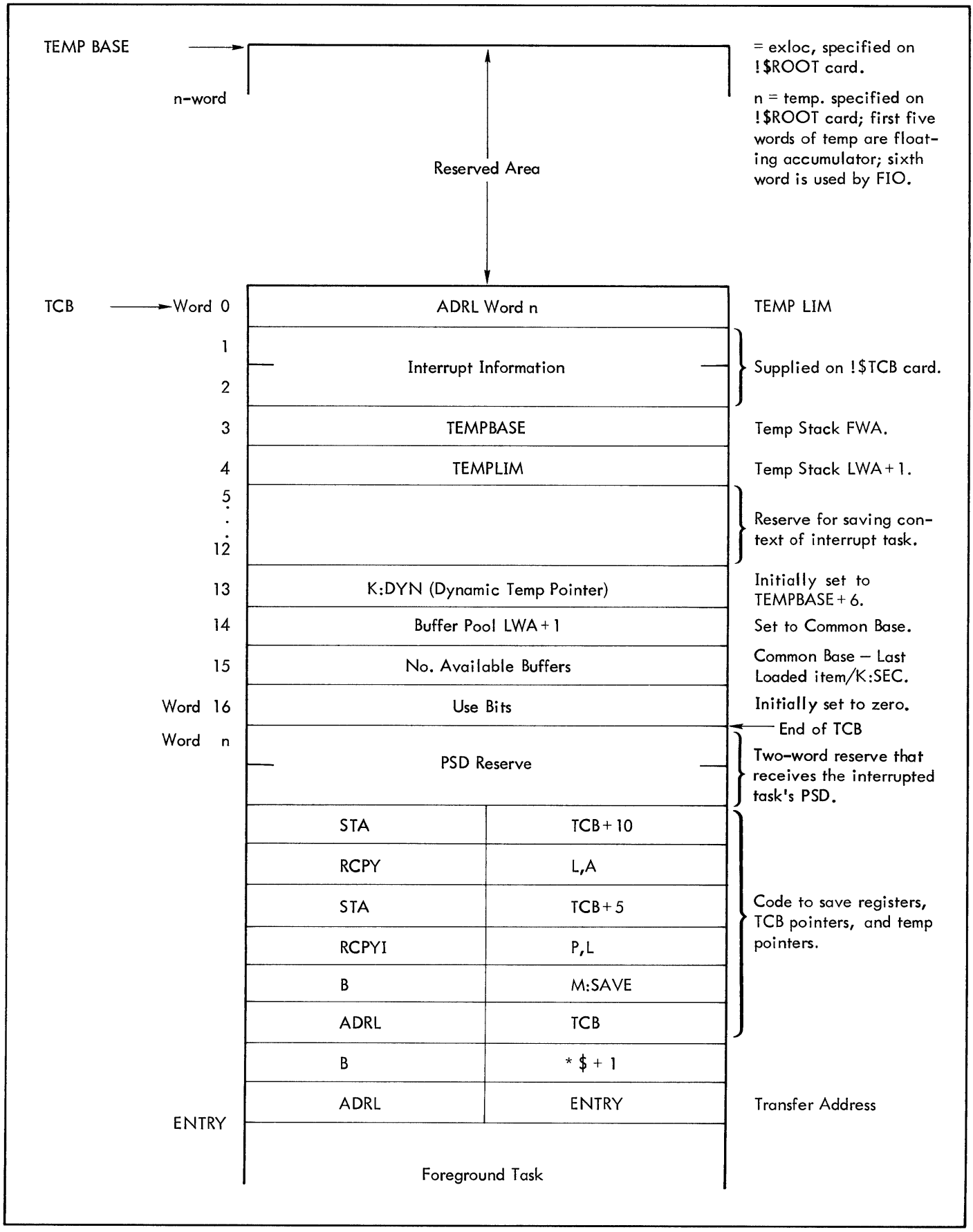


Figure 6. Task Entrance Format

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.

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 track boundary occurs, more than X'1FFF' bytes are requested, or a bad 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 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

```
LDS     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.

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 calling 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.

FOREGROUND CODING PROCEDURES

Conformity to the following conventions in coding foreground programs will increase the chances of recovery from a power failure:

1. Normally, when a task performs its own input/output, it will inhibit interrupts before it checks channel status, loads the channel registers, and issues the SIO. If the SIO instruction occurs within the 16 cells following the inhibit instruction, the Power On Task will be able to determine that I/O has not been initialized on this channel (if a power failure occurs after the interrupts are inhibited but before the SIO is issued) and therefore will not simulate an I/O interrupt.

2. If any task that uses the counter-equals-zero interrupt resets the dedicated interrupt location to zero before unloading or exiting, the Power Off Task will not arm and enable this level. This will prevent spurious interrupts.

3. For all Sigma 2 interrupts and the Sigma 3 external interrupts, the interrupt status is determined through the TCB chain (each TCB contains the address of the TCB of the task last interrupted). Therefore, any task that has entered its TCB in this chain will be reactivated. Entering the TCB chain is normally performed via a call to M:SAVE.

7. OVERLAY LOADER

The Overlay Loader can be used to create overlay programs 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 any number of 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 COMMON data area can also be established for use by the root and overlay segments.

Each segment is created by the 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 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 Loader accepts input in Standard Sigma 2/3 Object Language from predefined, prepositioned files, and prepares 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 ms, 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 Loader performs the following functions according to user options:

- Satisfies external reference/definition linkages and resolves forward reference and displacement chains.
- Searches specified libraries for unresolved references and loads these selected routines into core memory.
- Builds the OV:LOAD table for the loading of overlay segments.
- Writes the overlay cluster onto the OV file.
- Allocates COMMON.
- Allocates temporary storage stacks.
- Creates a Task Control Block (TCB) and initialization information.
- Creates the Public Library and associated transfer vectors (TVECT).
- Outputs maps of segment names and addresses, external definitions, and information concerning COMMON and temporary areas.

OVERLAY CLUSTER ORGANIZATION

The overlay cluster is the collection of absolute overlays formed by the Overlay Loader from relocatable binary object modules. (Note that the Loader does 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. Another 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 Loader. Note that at 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 above 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.

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.

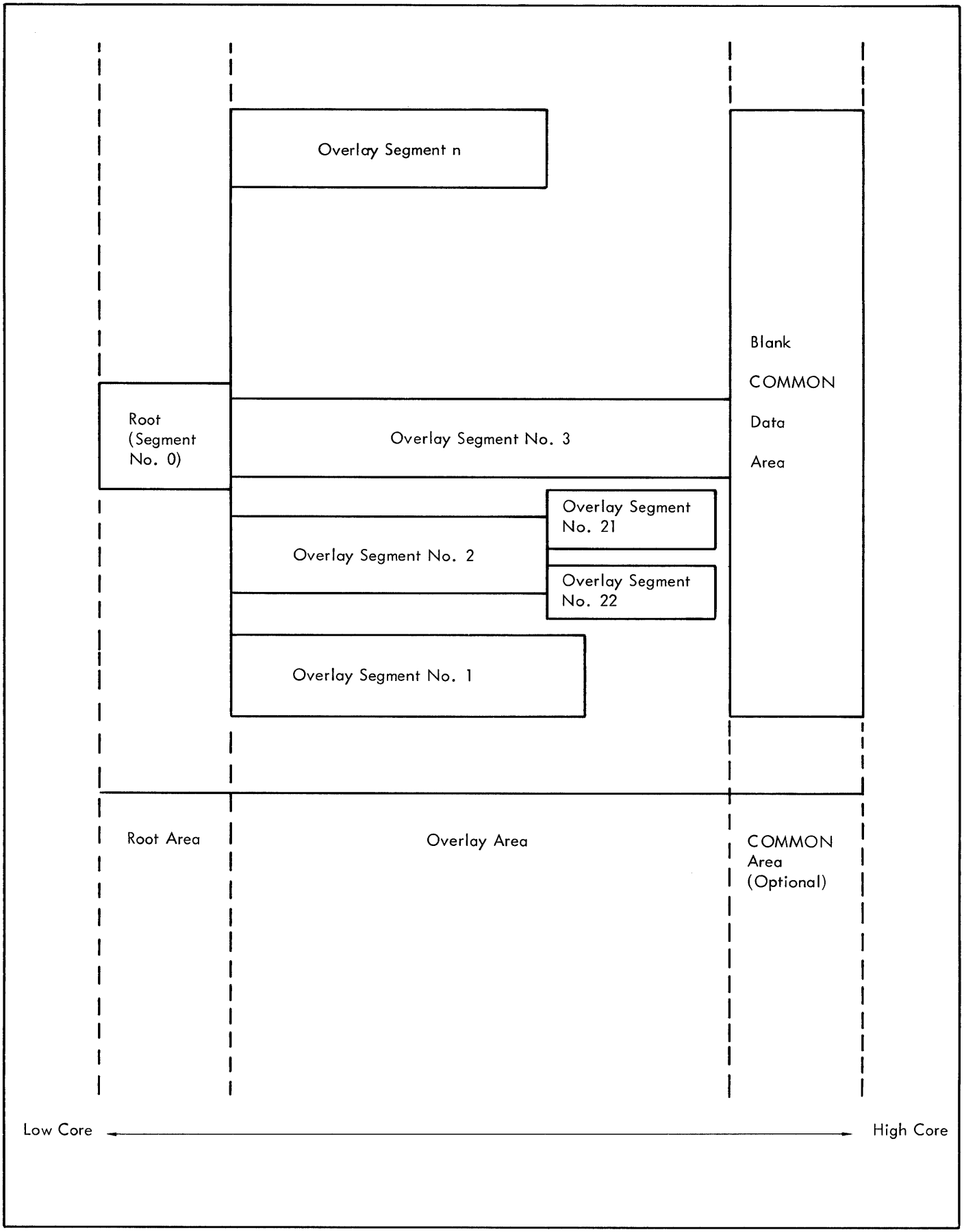


Figure 7. General Overlay Structure Example

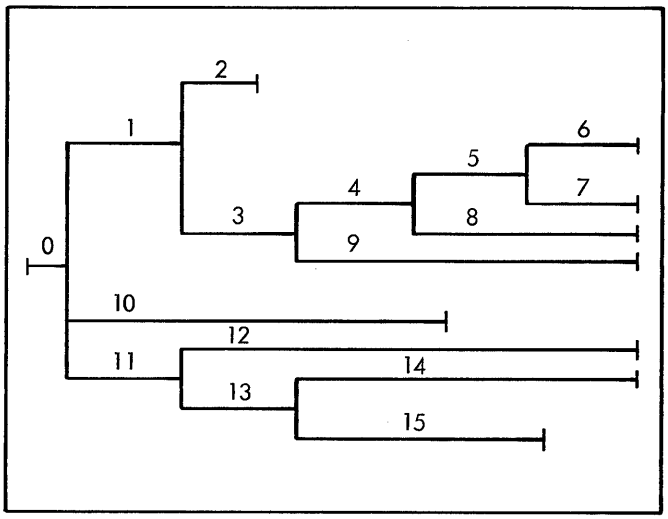


Figure 8. Sample Overlay Cluster Configuration

3. Because the Overlay Loader 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.

To satisfy any remaining unsatisfied primary references, the Overlay Loader searches the following libraries in the specified sequence:

1. Public Library
2. Monitor Service Routines
3. Basic or Extended Library
4. Main Library

CORE LAYOUT DURING LOADING

Background memory during the operation of the Overlay Loader is divided into four sections:

1. A fixed area large enough to contain the background temp stack, the Overlay Loader root, and the Loader overlays.
2. The segment table, fixed at 10(n + 1) where n equals the number of segments, which contains the user's OV:LOAD table.
3. A dynamic area in which the segment is loaded.
4. A dynamic area containing the symbol tables (allocation is 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 Loader references the operational labels listed below. Some assignments are user-defined, while others are handled internally by the Job Control Processor or by the Overlay Loader itself. All other operational labels referred to on !$LD cards must be assigned and positioned by the user prior to the !OLOAD card.

Label	Explanation
CC	Control commands.
DO	Control commands as read from CC, maps, and 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 Loader. Object modules are written onto GO by a preceding processor. The Loader rewinds GO initially. GO receives a default assignment by the Job Control Processor to the permanent file RBMGO in the System Data area.
LI	Assigned internally to System or User Library as library searches are performed.
OC	Abort messages and Overlay Loader messages that require operator attention.
OV	Output file for the Overlay Loader 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 Loader's 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.
RS	Assigned internally to read the RBM Symbol Table (RBMSYS) from the System Data area.
LS	Assigned internally to read the Public Library Symbol Table (LIBSYM) from the System Data area.
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 blocked 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 DO 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} ORG = xxxx  HLLOC = xxxx  CBASE = xxxx  CSIZE = xxxx  UMEM = xxxx  SECT = xxxx
              {BA}
*ROOT  ORG = xxxx  LWA = xxxx  LEN = xxxx  TRA = {NONE}  SEV = xxxx  OV:LOAD = xxxx
                                                {xxxx}
       [f1]  DEF  S1S2S3S4S5S6S7S8  L/I  S/U/P  B/E/M  yyyy
             :
             :
       [f1f2]REF  S1S2S3S4S5S6S7S8  LI  S/U     B/E/M  zzzz
             :
             :
*SEGMENT     IDENT   NODE   ORG   LWA   LEN   TRA   SEV
             xxxx    xxxx   xxxx  xxxx  xxxx  xxxx  xxxx
       [f1]  DEF  etc.
             :
             :
       [f1f2]REF  etc.
             :
             :
             REF
*SEGMENT
 :
 :
*SEGMENT
 :
 :
*ERRSEV = xxxx
*END MAP
```

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 COMMON.
CSIZE	Largest COMMON size encountered.

Figure 9. Load Map Format

Overlay Task Keywords (cont.)

UMEM	The number of locations between the end of the longest path, and either the beginning of 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. 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.
$S_1 \cdots S_8$	EBCDIC DEF/REF name of one to eight characters.
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 satisfied. For unsatisfied references, zzzz is blank.

Figure 9. Load Map Format (cont.)

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 (0 or 1). If any SEV > 0 or there are unsatisfied primary references, ERRSEV=1. Only in forming a PUBLIB do double DEFs cause ERRSEV=1.
END MAP	Completion of loading process.

Figure 9. Load Map Format (cont.)

CALLING OVERLAY LOADER

The Overlay Loader is requested via an !OLOAD 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]

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 COMMON size; for foreground tasks, cmn denotes either a base for COMMON or, in the case of zero COMMON, the upper limit of the task area.

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.

For foreground programs having COMMON, cmn denotes the base (i.e., FWA) of COMMON. In this case the effective upper limit of the program is cmn plus the largest

COMMON size allocation value encountered in loading. For foreground programs in which COMMON is allocated, but in which cmn has not been specified, the COMMON base is set by subtracting the first nonzero COMMON size allocation value encountered from K:BACKP-1. For foreground programs having no COMMON, cmn may be used to specify an upper limit for the program. If the program exceeds the limit, the Loader aborts. The default value of the area upper limit for foreground programs without COMMON is the upper limit of the nonresident foreground area (K:BACKP-1).

The Loader makes no distinction between programs loaded in resident and in nonresident foreground.

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 command, which is read by the Job Control Processor, the Overlay Loader control commands are read from the CC device under Loader control. 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 !$BLOCK control command defines operational labels that may require blocking buffers at run time. The list of such labels along with limits of available memory will be passed via the file header to M:LOAD, which will allocate a blocking buffer pool at run time. The pool will be utilized dynamically to provide blocking buffers in cases where a call to RBM routines M:READ or M:WRITE is not preceded by a call to M:OPEN. A call to M:CLOSE may release any such buffers. Thus, if two operational labels were to use a blocking buffer area at different times, the first might release the area for use by the second. Only one of the two labels would be required on the !$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 !$BLOCK command is allowed in a single job step. The format of the !$BLOCK command is

```
!$BLOCK oplb1,oplb2,...,oplbn
```

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 block file.

LIB The !$LIB control command specifies a new default library loading mode for the entire loading process. If the LIB command is not present, the Overlay Loader follows

the default case (Basic System Library). !$LIB cards may occur at any point in the control deck and will take effect from that point. The format of the command is

```
!$LIB  library,x[,y]
```

where

library must be one of the following EBCDIC codes.

Code	Library
B	Basic
E	Extended

x,y specify the order of search. The x and y parameters are either of the following EBCDIC codes.

Code	Library
S	System
U	User

The order in which they are specified determines the order of search. Note that if y is not specified, only x will be searched.

MS ML MP The MAP control commands specify that map information is to be output on DO. 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

```
!$MS
```

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, respecitvely. 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 !$TCB control command indicates (for a foreground task only) that the Overlay Loader must create a TCB and reserve a PSD location, and must generate a call to RBM routine M:SAVE. In addition, information to initialize the TCB at run time will be passed in the file header. If no !$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. The !$TCB command must precede the !$ROOT command. The format of the command is

```
!$TCB w1,w2
```

where w_i are the values to be placed in words 1 and 2 of the created TCB. (See Chapter 6, Real-Time Programming.)

The Overlay Loader will handle specific and default cases of program execution and TCB initialization within the framework of the following restrictions:

- The Overlay Loader defines 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 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 !$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.

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, !$INCLUDE, !$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]
```

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.

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. No default is provided.

Note that if the oplb parameter is absent, !$LD (Load) or !$INCLUDE control commands must follow !$ROOT to specify loading. If oplb is present and the n parameter is empty, loading proceeds from oplb until an !EOD 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]
              [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 empty or is an ident, one module is loaded.

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 library,m[,n]
```

where

library must be one of the following EBCDIC codes:

Code	Library
B	Basic
E	Extended

m[,n] specify the order of search. The m and n parameters are either of the following codes:

Code	Library
S	System
U	User

If n is not specified, only m will be searched. There are no default cases for E, B, m, and n.

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.

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.

$value_i$ 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". Note that it is not possible to modify a library module by use of an !$MD control command.

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 16,112 bytes since the segment and its library are written as one record on the RAD.

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 !$SEG control command. The control command stack is terminated by an !EOD. The user may not defer library loading to a higher level segment. The Loader will attempt to satisfy all references present at a level from the libraries specified on !$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]
```

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.

The following rules should be observed in defining segments for the overlay cluster:

1. 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.

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.

At execution time an explicit call to RBM routine M:SEGLD with the segment identifier number and the ADRL OV:LOAD causes the reading of that segment into memory from the OV file. Thus, any segment may, by an explicit call, cause any other segment to be loaded for execution.

PUBLIB The !$PUBLIB control command indicates that the Overlay Loader is to create a Public Library using modules that follow and/or modules from selected libraires. The Public Library is biased at the location specified in K:PLFWA of the RBM. Each symbol is flagged as Extended, Basic, or Main according to control information on the !$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. To conserve core space when forming the Public Library, the Loader will remove any trailing RES from a library routine and will also change the appropriate word in the calling sequence for M:RES, M:PUSH, or M:PUSHK so that the dynamic temporary stack will be used for temporary storage space.

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 8191 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. If an operator response is required, the message

```
!!BEGIN WAIT
```

is written on OC and DO. The operator activates the console interrupt and keys in either of the following codes.

Code	Meaning
S	Continue.
X	Abort Overlay Loader and return control to RBM.

The format of the error message where an operator response is required is

```
OLERR xx
```

where xx is a two-letter mnemonic that identifies the error.

The types of Overlay Loader messages are as follows:

1. Warning messages, after which loading continues.
2. Response messages, requiring an S or X key-in from the operator.
3. Abort messages, upon which the Overlay Loader exits via the RBM routine M:ABORT (see Appendix C for abort codes, abort messages, and their meanings).

The Overlay Loader error messages are given in Table 18 below.

Table 18. Loader Error Messages

Message	Meaning
LIBSYM UNDEFINED[t]	There was no file entry on the System Data area of the RAD or disk pack for the LIBSYM table.
OLERR CC !!BEGIN WAIT	A control command card has a format or parameter error. An S response causes the next control command to be read in from CC. This may be a corrected command to replace the one in error.[tt]
OLERR CS !!BEGIN WAIT	There was a checksum error on a binary record. An S response causes the record to be reread.
OLERR IB !!BEGIN WAIT	Illegal binary format (that is, the first word was not 'FF' or '9F') was detected. An S response causes the record to be reread.[tt]
OLERR ID !!BEGIN WAIT	The indent on the binary module just loaded does not compare with the ident specified on the !$LD command. On an S response, the Loader accepts the binary module as is and continues processing.
OLERR IS !!BEGIN WAIT	Control commands were improperly sequenced in the control command stack. An S response causes the next control command to be read. However, if the sequence error was due to a SEG command, the Loader aborts.[tt]
OLERR SQ !!BEGIN WAIT	There was an incorrect sequence number on a binary record. An S response causes the record to be reread.[tt]
OLERR TA	No transfer address was encountered in the loading of the root segment. This is only a warning message. The Loader sets a default transfer address as the first word of the program.
OLERR UR	There were unsatisfied references in the path. This in only a warning message.
TOO MANY DEFS[t]	There were more DEFs in the Public Library than were allocated at system generation.

[t]This message may be written on DO during writing of the Public Library, LIBSYM, or TVECT table onto the RAD or disk pack. 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 RAD, disk pack, 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.

8. RAD EDITOR

INTRODUCTION

The RAD Editor controls RAD and disk pack allocation by generating and maintaining directories for all permanent files. Through control command input, the RAD Editor can

- Add or delete entries in permanent file directories.
- Copy data from one file into another.
- Maintain library areas on RADs or disk packs for use by the Overlay Loader.
- Copy an object module contained in a library.
- Map file allocation.
- Dump contents of random-access files.
- Save the contents of RADs or disk packs in self-reloadable form.
- Clear any permanent area.
- Skip bad tracks when allocating a file area.

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 Dn)

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.

PERMANENT RAD/DISK PACK AREA ORGANIZATION

Every permanent area has its own directory that begins in the first sector of the area. The first entry contains the address (if any) of the bad tracks within the area. Each succeeding directory entry indicates the name, length, location, and format of a file in the permanent area. Directories are linked; that is, after a sector of a directory is filled, the next available sector within the permanent area is allocated as the continuation of the directory.

The permanent file directories are software write-protected. There are four levels of write protection: no protection, write permitted by RBM only, write permitted by foreground, and write permitted by background. Write protection for files is a user option. Therefore, an SY key-in must be initiated before updating or initializing a file directory, updating any protected file, or copying data into a protected file.

Space with an area is allocated sequentially, and tracks designated as bad are skipped at file allocation. The first file in the area begins in the second sector and extends over an integral number of sectors. Thus, every file begins and ends on a sector boundary. 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 to the designated directory. Control commands allow directory entries to be added or deleted.

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 Loader 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).

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.

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 COMMON, extended-precision, and single-precision routines).
- 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).
- 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 COMMON library.
- r_j is the number of REFs in the jth library routine in the COMMON library.
- d_j is the number of DEFs in the jth library routine in the COMMON library.
- g is the granule size in words.

The number of granules in the MODULE file is

$$MODULE_n = \sum_{i=1}^{n} \frac{60}{g} (c_i)$$

where

n is the number of modules in the library (including COMMON, extended-precision, and single-precision routines).

g is the granule size in words.

c_i is the number of record images in the ith library routine.

RAD EDITOR OPERATIONAL LABELS

The RAD Editor requires the operational labels listed below for input/output. These labels are reserved for use by the RAD Editor and must not be used on !#DUMP or !#FCOPY commands.

The following labels must be assigned before requesting the RAD Editor:

Label	Explanation
BI	Object module input to System and User Library
BO	Output of copies of object modules from the System and User Libraries.
CC	Control command input.
DO	Log of control commands, error messages, and operator key-ins.
LO	Maps of directories and dumps of files.
OC	Messages to the operator and key-ins from the operator.
X1-X6	Assigned and used internally by RAD Editor for RAD maintenance.

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. The form of the command is

```
!EOD
```

CONTROL COMMAND FORMAT

All RAD Editor control commands are input from CC and listed on DO. If CC and DO are assigned to the same device, the commands are not listed. 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 32,768, 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 embedded 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.

CONTROL COMMAND REPERTOIRE

ADD The !#ADD command adds a new entry to the specified permanent file directory. It defines the name, write-protection, format, and length of a new file. Adding an entry to a directory causes space to be allocated for

the new file. Once space has been allocated, data can be written on the file. The form of the command is

```
!#ADD directory,name,file [,record] [,format]
    [,write] [,foreground]
```

where

directory specifies a permanent file directory (i.e., not BT or CP). It must be a currently defined area.

name is the file name. The file name is composed of three to eight EBCDIC characters. If a file contains a processor that is loaded with a processor command (!name), the name of the file must be identical to the one used in the command. Before using the !#LADD, !#LREPLACE, and !#LDELETE commands (explained below), entries for six special files in the library area (SL or UL) must be added. The codes for the library files must be one of the following:

Code	File
MODIR	Module Directory
EBCDIC	EBCDIC
EDFRF	Extended DEF/REF
BDFRF	Basic DEF/REF
MDFRF	Main DEF/REF
MODULE	Module

For data files, name is determined by the name parameter.

file is the number of records in the file, and must be either a hexadecimal value or a decimal integer.

record is the maximum number of bytes per logical record and may not exceed 32,168 bytes. "Record" may be expressed as either a hexadecimal value or a decimal integer. The meaning of "record" is determined by the format parameter. The record parameter need not be input for library or processor files,† since these have predefined record lengths. For blocked sequential-access files, logical record size is 120 bytes by default. For blocked compressed sequential-access files, the logical record size is 80 bytes by default. For unblocked sequential-access files, logical record size is the sector size of the device by default. For random-access files, "record" is the granule size. The default granule size is the sector size of the device, and for random files, granule size is the sector size of the device by default.

format specifies the structure of the file. It must be one of the following codes:

Code	Format
B	Blocked sequential-access file
C	Blocked compressed sequential-access file
R	Unblocked random-access file
U	Unblocked sequential-access file

If omitted, the format parameter is B for data files and R for all library or processor files.†

write specifies write-protection for the file. It must be one of the following codes:

Code	Directory
B	Write permitted from background
F	Write permitted from foreground
N	No write protection
R	Write permitted from RBM only

If omitted, the write parameter is N for all files.

foreground is applicable only if "directory" is UP. If "foreground" is F, the named file contains a resident foreground task. If "foreground" is N or omitted, the file does not contain a resident foreground task.

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

```
!#DELETE directory,name
```

where

directory specifies a permanent file directory. It must be a currently defined file.

†For allocating files, if the last character of "directory" is P, the file is a processor file by default; if the character is D, the file is a data file by default.

name is the file name for the entry to be deleted. The file name is composed of a maximum of eight EBCDIC characters, in which at least one character is alphabetic.

FCOPY The !#FCOPY (File Copy) command copies data from one random-access file to another. The file copy process terminates 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 random-access RAD file. The COPY Utility Routine (see Chapter 10) must be used to copy sequential-access files.

$oplb_1$ is the input file.

$oplb_2$ is the output file.

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 a sequential file. The object module on BI must be in Standard Sigma 2/3 Object Language. Any blank card or binary card on BI that contains only zeros is ignored. The form of the command is

```
!#LADD directory [,identification] ,library
```

where

directory specifies a permanent file directory. It must be one of the following codes:

Code	Library
SL	System
UL	User

identification is the program name located in the start module item of the object module on BI. Within a permanent area (SL or UL), each object module must have a unique "identification". 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.

library specifies the target library. It must be one of the following codes:

Code	Library
B	Basic Library (single-precision math library routines)
E	Extended Library (extended-precision math library routines)
M	Main Library (nonmath library routines)

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 Sigma 2/3 Object Language. Any blank card or binary card (on BI) that contains only zeros is ignored. The form of the command is

```
!#LREPLACE directory,identification,library
```

where

directory specifies a permanent file directory. It must be one of the following:

Code	Library
SL	System
UL	User

identification 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.

library specifies the target library. It must be one of the following codes:

Code	Library
B	Basic
E	Extended
M	Main

LDELETE The !#LDELETE (Library Delete) command deletes an object module from the designated library. The form of the command is

```
!$LDELETE directory,identification,library
```

where

directory specifies a permanent file directory. It must be one of the following:

Code	Library
SL	System
UL	User

identification is the program name of the object module to be deleted.

library specifies the target library. It must be one of the following codes:

Code	Library
B	Basic
E	Extended
M	Main

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 directory,identification
```

where

directory specifies a permanent file directory. It must be one of the following:

Code	Library
SL	System
UL	User

identification 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 directory
```

where directory specifies either User Library (UL) or the System Library (SL).

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. Maps of library directories include program name, library designation (Main, Basic, or Extended), and DEFs and REFs for each object module. The form of the command is

```
!#MAP [directory1] [,directory2]... [,directory6]
```

where

$directory_i$ specifies a file directory. It must be a currently defined area. As many as eight directories may be input.

If no directory parameter is included, all currently defined directories are mapped.

DUMP The !#DUMP command dumps a random-access file on LO. The file is dumped one granule at a time. The DUMP Utility Routine (see Chapter 9) may also be used to dump sequential-access files. DUMP represents each word as a four-character hexadecimal number. It dumps each granule of the file starting at BOT (if starting address is not specified) and ending at EOT or after the specified number of granules has been dumped. The form of the command is

```
!#DUMP  oplb[,number 1][,number 2]
```

where

oplb is the operational label or FORTRAN device unit number (e.g., F:109) of a temporary, checkpoint, or permanent RAD file to be dumped.

number 1 is the starting granule address in decimal or hexadecimal.

number 2 is the number (decimal) of granules to be dumped. If the number parameter is omitted, the file is dumped up to EOT.

SAVE The !#SAVE command saves the contents of the RAD for subsequent restoration. The image of the designated permanent area (including both directory and files) and the RBM bootstrap are written on magnetic tape BO in a self-reloadable format; the BO output contains a bootstrap loader followed by the RAD images of the RBM bootstrap and the designated area. Execution of the bootstrap loader causes the RAD image to be read into memory and restored onto the RAD without RBM control; after restoration, the RBM bootstrap is executed. The BO output can also be restored on the RAD via the !#RESTORE command (explained below). The form of the command is

```
!#SAVE [directory1] [,directory2]...[,directory6]
```

where

directory$_i$ specifies the permanent RAD area to be saved. It must be a currently defined area.

If no directory parameter is included, all current permanent file areas are saved. A request to save CP or BT is ignored.

RESTORE The !#RESTORE command restores the permanent areas saved via a !#SAVE command. It reads the output of the !#SAVE command from BI and bypasses the bootstrap loader. The form of the command is

```
!#RESTORE
```

SQUEEZE The !#SQUEEZE command compacts the designated file areas. Unused space is regained by regenerating the dictionaries and moving files. The form of the command is

```
!#SQUEEZE [directory1] [,directory2]... [,directory6]
```

where

directory$_i$ specifies the permanent RAD area to be compacted. It must be a currently defined area.

If no directory parameter is included, all current permanent areas are compacted. A request to squeeze CP or BT is ignored.

CLEAR The !#CLEAR command zeros out the specified RAD area or file. The form of the command is

```
!#CLEAR directory [,file]
```

where

directory specifies a permanent area. It must be a currently defined area.

file is a file name, within the area specified by "directory" which is to be cleared. The remainder of the area will be unchanged. If "file" is omitted, the entire area is cleared. Note that only one area may be cleared with each !#CLEAR command.

TRACKS The !#TRACKS command will update the list of bad tracks for each RAD or disk pack device. This list resides in the first sector of each area. The source for this update is the existing Alternate Track Pool, which is modified via the BT key-in. The form of the command is

```
!#TRACKS
```

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 outputs error messages on OC and DO. If OC and DO are assigned to the same device, duplication of messages on DO is suppressed. If an operator response is required, the message

```
!!BEGIN WAIT
```

is written on OC and DO. The operator activates the console interrupt and keys in either of the following codes.

Code	Meaning
S	Continue.
X	Abort RAD Editor and return control to RBM.

To abort, the RAD Editor calls the Background Abort routine, M:ABORT. If the RAD Editor aborts because of an irrecoverable input/output error, the code in the abort message is the operation label of the device in error. If the abort is due to an X response by the operator or some error condition, the code is 'RE'.

The error messages output by the RAD Editor and their meanings are given in Table 19. The messages in Table 20 are written on the keyboard/printer during RAD restoration via the bootstrap loader produced by SAVE. Any error output causes the computer to go into a wait state after writing the appropriate message.

Table 19. RAD Editor Error Messages

Message	Meaning
AREA OVERFLOW	Allocation of the amount of storage indicated by the file parameter on the !#ADD command would cause the permanent area indicated by the directory parameter to overflow. RAD Editor reads the next command from CC.
ASSIGN ERR	The RAD Editor was unable to assign an operational label to a file because the number of available RAD or disk pack device-file numbers is insufficient. RAD Editor aborts.
BOT oplb	An unexpected beginning-of-tape has been encountered on the device having the operational label oplb. RAD Editor aborts.
CKSM ERR	The last record in the object module being read from BI has a checksum error. If the job is ATTENDed, operator response is solicited; an operator response of S causes the Editor to read the next record from BI. RAD Editor aborts.
CORE OVERFLOW	The last command cannot be processed for lack of background space. The RAD Editor aborts.
DUP IDENT	The last object module read from BI cannot be added to the library with a !#LADD command because it is already in the library. RAD Editor aborts.
DUPLICATE NAME	An attempt was made to add a file whose name already exists for this area. The RAD Editor reads the next command from CC.
EDIT ERR	Data on the RAD or disk pack has been rendered invalid. RAD Editor aborts.
EOF oplb	An unexpected end-of-file was encountered on the device having the operational label oplb. RAD Editor aborts.
EOF READ FILE	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	An unexpected end-of-tape was encountered on the device having the operational label oplb. RAD Editor aborts.
EOT WRITE FILE	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 from CC.

Table 19. RAD Editor Error Messages (cont.)

Message	Meaning
ERR I/O oplb	A calling sequence error occurred for input/output on the device having the operational label oplb. RAD Editor aborts.
FILE OFLO	A file in the library area has overflowed during execution of a !#LADD command. If operator response is S, the next command is read.
ILLEG BIN	An illegal binary record (first byte not X'FF' or X'9F') has been read with an object module on BI. RAD Editor aborts.
INV CTRL	Control command is invalid. It cannot be recognized by RAD Editor or has incorrect syntax. If operator response is S, the next command is read.
INV I/O OP oplb	An invalid input/output operation was attempted on the device having the operational label oplb. RAD Editor aborts.
LENGTH ERR oplb	A record of incorrect length was read from or written on the device having the operational label oplb. RAD Editor aborts.
LOAD ERR	The RAD Editor overlay cannot be loaded. RAD Editor aborts.
NO BLOCK oplb	No blocking buffer is available for the file assigned to the operational label oplb. RAD Editor aborts.
NO IDENT	The object module on BI does not have the same "identification" in the start module item as indicated on the !#LADD command, the identification in start module item is blank, or there is no object module on BI. RAD Editor aborts.
NONEXISTENT FILE	An attempt was made to delete a file whose name does not exist in the specified area. The RAD Editor reads the next command from CC.
PARAM ERR	Control command has a parameter error. A parameter has incorrect content, has been omitted, or is not consistent with the other parameters. A parameter error also occurs for duplicate Editor commands; that is, when an already-existing file is created via the !#ADD command or when a nonexisting file is deleted via the !#DELETE command. If operator response is S, the next command is read.
RE ERR	RAD could not be restored completely because either BI input is out of sequence, or permanent RAD areas in the Master Directory do not agree with BI input. RAD Editor aborts.
SEQ ERR	The last record in the object module being read from BI has a sequence error. If the job is attended, an operator response of S causes the Editor to read the next record from BI. If the job is not attended, RAD Editor aborts.
SZ ERR	The object module on BI cannot be placed in the library because it has more than 61 external definitions and references. RAD Editor aborts.

Table 19. RAD Editor Error Messages (cont.)

Message	Meaning
UNPROTECT RAD	The RAD or disk pack is write-protected. RAD Editor continues to attempt writing. The operator should interrupt and key in SY, reset the appropriate RAD protection switches, or interrupt and key in X to abort, whichever is appropriate.
UNRECOVER I/O oplb	An irrecoverable I/O error occurred on the device assigned to the operational label oplb. RAD Editor aborts.
WRITE PRO oplb	The magnetic tape assigned to the operational label oplb is write-protected. RAD Editor aborts.

Table 20. RAD Restoration Messages

Message	Meaning
CHCK WRITE ERR	A check write error occurred (that is, data recorded on the RAD or disk pack could not be verified).
CHECKSUM ERR	The last record image read has a checksum error.
RAD WRITE PRO	The RAD or disk pack is write-protected.
READ ERR	The last record being read had a read error.
RESTORED VXX	RBM version XX has been restored on the RAD.
SEQUENCE ERR	The record images for restoration are out of sequence.

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[t]).
- Generate or update files that contain Standard Sigma Object Language modules (Object Module Editor).
- Generate or update symbolic files (paper or magnetic) that contain source data (Record Editor[tt]).
- 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.

[t]Dump is known as Paper Tape Dump in the BCM system.

[tt]Record Editor is known as Paper Tape Editor in the BCM system.

UTILITY EXECUTIVE PROGRAM

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 device (SI).

If a specific Utility subroutine is requested, the Program Executive verifies that the subroutine is in core 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 and listed on the DO 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
!!UKEYIN
```

The operator response, either an S to continue or an X to abort, determines whether or not the Utility program continues.

If the response is S, the Source Input Interpreter reads the next control command from SI. 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
!!UKEYIN
```

The operator response, either an S to continue or an X to abort, determines whether or not the Utility program continues.

OPERATOR COMMUNICATION ROUTINE

All messages to the operator are written on the OC device by the Operator Communication routine.

If a response is required from the operator, the Operator Communication routine types the following message:

```
!UKEYIN
```

The operator then keys in either an S to continue, or an X to abort.

If the response is S, a return is made to the calling routine. If the operator keys in an invalid response (not S or X), the following message is written on OC and DO.

```
 KEY ERR
!!UKEYIN
```

The operator then types in the correct response.

INPUT/OUTPUT ERROR MESSAGES

The Program Control routine performs all input/output checking for the Utility program. Messages regarding input/output errors are written on both the OC and DO devices.

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 RBM control command input, Utility program control commands, and various modification source inputs.
DO	Device for listing of control commands (as they are interpreted), messages, error conditions, operator responses, etc. DO provides a permanent log of the control command flow. This is the only operational label for the Program Control routine that can be assigned to device-file number 0 (i.e., suppressed). If OC and DO are assigned to the same device, duplication of messages is suppressed.
OC	Device for messages to the operator, or key-in responses from the operator (always via the keyboard/printer).
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 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: when the !UTILITY command has no name parameter, the !*PRESTORE control command allows the user the option of prestoring SI 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 a !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 Executive or 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.

CONTROL COMMAND FORMAT

All Utility program control commands are input from SI and are listed on the DO 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 command 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]
```

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]
```

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
!!UKEYIN

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 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.

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

```
!*SKIP oplb[, number]
```

If oplb is assigned to a blocked sequential-access RAD file, the number parameter is the number of logical records to skip.

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
```

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.

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 file (blocked, unblocked, or compressed) to cards, paper tape, magnetic tape, line printer, keyboard/printer, and sequential-access RAD file (blocked, unblocked, or compressed). Using control functions of the Control Function Processor, records and files can be skipped. Output generated by the COPY routine can be verified. If the binary mode is requested for either copying or verifying, file marks are recognized for a magnetic tape or sequential-access RAD file.

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 BIN control card 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 9T magnetic tape 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 only be used to copy nonstandard binary records. Since no editing is done when a binary read operation is specified, NL, EOM, and ¢ are not interpreted as editing characters. All records are copied on a byte-for-byte basis (including leading and trailing blanks). EOD is not recognized as a file mark. Therefore, a request to copy/verify one or more files causes input to terminate only when the input device goes into manual mode. A request to copy/verify one or more times (when the input device is magnetic tape) is processed normally, since file marks are recognized.

COPY OPERATIONAL LABELS

The following operational labels are used by the COPY routine in addition to the Utility subsystem operational labels:

Label	Device
UI	Input device
X4	Verify device

Other operational labels may be used by COPY (at the option of the user) to specify the input and output devices for verifying and copying, respectively.

COPY OPERATING CHARACTERISTICS

The COPY routine checks whether input/output operational labels are assigned to the same physical device. 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, the message

```
LD INPUT

!!UKEYIN
```

is written on the OC and DO device, and the Operator Communication routine waits for an operator response. The operator should load the input at this point and key in an S response 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.

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 comparision for VERIFY requests, and must follow the !UTILITY command. The input for COPY operations is read from UI. For VERIFY operations, X4 is read. UI may not be used as a parameter for COPY operations; nor are UI and X4 allowed as parameters for VERIFY operations. An !*OPLBS command should follow !*ASSIGN commands that change the device type of UI or X4. Operational

labels may be assigned to any device except a random-access RAD file. Assignments remain in effect until a new !*OPLBS command is read. The form of the command is

```
!*OPLBS oplb1[,oplb2 ...][,oplbn]
```

where $oplb_i$ is the optional label for an output device for subsequent !*COPY commands, or an input device for subsequent !*VERIFY control commands. The oplb parameter may not be assigned to device-file number 0 ($n \leq 8$).

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]
```

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.

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]
```

The parameters are defined as for the !*COPY control command.

Note: UI must not be used on an !*OPLB command with VERIFY.

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 or the EBCDIC 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 BIN 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
SI	Device for input commands.
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

!!UKEYIN
```

is written on the OC and DO device. The operator mounts the input tape and keys in an S response 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 DO and dumping is performed.

CALLING DUMP

The DUMP routine is requested with the control command

```
!UTILITY DUMP[, oplb]
```

where oplb is the operational label of the input devoce. If oplb is omitted, the operational label is assumed to be UI.

DUMP CONTROL COMMANDS

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. If the dump "input" (UI) is assigned to a sequential-access RAD file, the number parameter must be 1.

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, EBCDIC is assumed.

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 Standard Sigma 2/3 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 Sigma 2/3 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 from which binary object modules are to be inserted.
LO	Device for listing either UI or UO object module names.

Label	Explanation
UI	Input device.
UO	Output device.

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. If so, the message

```
LD LIST

!!UKEYIN
```

is written on OC. The operator responds by preparing UI and keying in an S response. Listing of the modules proceeds.

If no two of the labels SI, BI, or UI are assigned to the same device, control commands on SI 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 SI 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.

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	Prestored Data
SI, BI	SI
SI, UI	SI
BI, UI	BI
SI, BI, UI	SI, BI

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, 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

   !!UKEYIN
   ```

   is written on OC and DO. The operator loads the modules to be inserted on the BI device and keys in an S response. If SI and BI are assigned to different devices, 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

   !!UKEYIN
   ```

   is written on OC and DO. The operator must prepare UI and key in an S response.

3. The mode modification control commands are interpreted, causing updating or generation to proceed. Each control command is listed on DO as it is interpreted.

If no two of the operational labels SI, BI, and UI are assigned to the same device, 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 read from SI is listed on DO.

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 SI 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
```

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 or !*LIST control command is interpreted from SI, or two !EODs from BI or UI. 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 physical 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.

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 program name (up to eight EBCIDC 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. 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.

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, the following message is written on OC and DO:

MODE ERR UI,device

!!UKEYIN

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 from the SI device. Each !*LIST control command file mark, or !EOD causes the line numbering to restart with 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 UI is assigned to a sequential-access RAD file, the number parameter must not be greater than 1.

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 number$_1$[, number$_2$]

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).
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 SI, 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 recommended that files be sequenced as they are generated to avoid an update pass at a later stage. The user can generate one file (terminated by an !EOD or an !*END from SI) wherein a single file mark is written on UO, or multiple files (terminated by two !EODs or !*ENDs from SI) 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.

Files can be sequenced as they are generated or updated. 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 updated 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 indicates that SI sequence errors are to be ignored if UO is being generated or that UI sequence errors are to be ignored if UI is being updated. If IGN is used, no sequence error messages are printed.

ALL indicates that the GEN function is to continue until two !EOD or !*END cards are encountered from SI, or that the update function is to continue until two EOFs are encountered from UI.

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, device

!!UKEYIN
```

SEQUENCE EDITOR 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 from UI and the message "DELETED", followed by the card image from SI and the message "INSERTED" are output on LO.

Insertion. 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

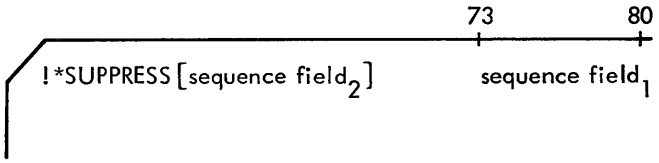
```
                                                    78        80
!*DELETE [sequence field2]                          sequence field1
```

where

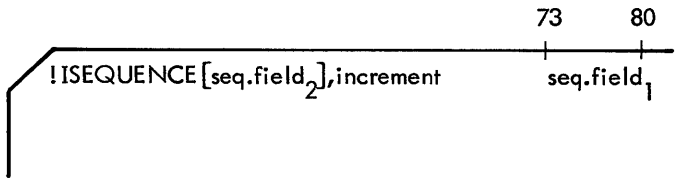
sequence field$_2$ indicates that the images are to be deleted from the ident and/or sequence number in sequence field$_1$ up to and including the ident and/or sequence number in sequence field$_2$.

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 card images are listed on LO. The form of the command is



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 command takes effect with the next card image to be written on UO.

UTILITY ERROR MESSAGES

Unless otherwise noted, the following definitions apply in error messages given in Tables 21 through 26:

Code	Explanation
oplb	Operational label of the device.
device	Device type or physical device number.

The operator response to !!UKEYIN is

Code	Meaning
S	Continue
X	Abort

When an irrecoverable error occurs, the Utility program aborts. For an irrecoverable input/output error on OC or DO, the code in the abort message is the operational label of the device. For other operational labels, the irrecoverable input/output message is written. Abort returns, due either to error or X operator responses, cause UT to appear in the abort message.

Table 21. I/O Error Messages

Message	Meaning
BOT oplb, device !!UKEYIN	An attempt has been made to backspace over the magnetic tape load point or the beginning-of-tape of a RAD file.
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 the RAD (may occur with BCD 7-track tapes). See M:READ status returns in Chapter 4 of this manual.
EMPTY oplb, device !!UKEYIN	Manual intervention is required (the device is in the manual mode or no device is recognized).
EOF oplb, device !!UKEYIN	An unexpected tape mark, end-of-file (RAD), or !EOD has been read from magnetic tape, cards, paper tape, keyboard/printer, or RAD file.
EOT oplb, device !!UKEYIN	The end-of-tape mark on a magnetic tape or RAD file has been sensed.

Table 21. I/O Error Messages (cont.)

Message	Meaning
IL RAD SEQ oplb, device !!UKEYIN	An operational label was assigned to a random-access RAD file, or an attempt was made to skip, read, or write more than one RAD file.
INV I/O OP oplb, device !!UKEYIN	An input/output operation is not meaningful for the requested device.
INV OPLB oplb, device !!UKEYIN	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.
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. The Utility program aborts.
UNRECOV I/O oplb, device !!UKEYIN	An irrecoverable input/output error has occurred after the maximum number or retries has been unsuccessfully attempted.
WRITE PRO oplb, device !!UKEYIN	An attempt has been made to write on a write-protected magnetic tape or RAD file.

Table 22. Control Function Command Error Messages

Message	Meaning
FSKIP Command	
DEOF oplb, device !!UKEYIN	Two consecutive file marks were encountered before the required number of files had been passed.
EOT oplb, device !!UKEYIN	The end-of-tape was encountered before the required number of files has been passed.
IL RAD SEQ oplb, device !!UKEYIN	The number parameter is not 1 and "oplb" is assigned to a sequential-access RAD file, or the oplb parameter is assigned to a random-access RAD file.
INV OPLB !!UKEYIN	The operational label identifies an invalid device.
PARAM ERR !!UKEYIN	The oplb parameter is missing, or the number parameter is nonnumeric or greater than 32, 767.
RSKIP Command	
EOF oplb, device !!UKEYIN	An !EOD or file mark was encountered before the required number of records was passed.
EOT oplb, device !!UKEYIN	An end-of-tape was encountered before the specified number of records was skipped.
IL RAD SEQ oplb, device !!UKEYIN	The oplb parameter is assigned to a random-access RAD file.
INV OPLB !!UKEYIN	The oplb parameter identifies an invalid device.
PARAM ERR !!UKEYIN	The oplb parameter is missing, or the number parameter is nonnumeric or greater than 32, 767.

Table 22. Control Function Command Error Messages (cont.)

Message	Meaning
FBACK Command	
BOT oplb, device !!UKEYIN	The beginning-of-tape was encountered before the required number of files had been passed.
DEOF oplb, device !!UKEYIN	Two consecutive file marks were encountered before the required number of files was backspaced.
IL RAD SEQ oplb, device !!UKEYIN	The oplb parameter was assigned to a random-access RAD file.
INV OPLB oplb, device !!UKEYIN	The operational label identifies an invalid device.
PARAM ERR !!UKEYIN	The operational label parameter is missing or contains more than two characters, or the number parameter is nonnumeric or greater than 32,767.
RBACK Command	
BOT oplb, device !!UKEYIN	The beginning-of-tape was encountered before the requested number of records had been passed.
EOF oplb, device !!UKEYIN	A file mark was encountered before the requested number of records had been passed.
IL RAD SEQ oplb, device !!UKEYIN	The oplb parameter was assigned to a random-access RAD file or a compressed EBCDIC RAD file.
INV OPLB oplb, device !!UKEYIN	The operational label identifies an invalid device.
PARAM ERR !!UKEYIN	The operational label parameter is missing or contains more than two characters, or the number parameter is nonnumeric or greater than 32,767.
REWIND Command	
IL RAD SEQ oplb, device !!UKEYIN	The oplb parameter is assigned to a random-access RAD file.
PARAM ERR !!UKEYIN	The oplb parameter contains more than two characters.
UNLOAD Command	
IL RAD SEQ oplb, device !!UKEYIN	The oplb parameter is assigned to a random-access RAD file.
INV OPLB oplb, device !!UKEYIN	The oplb parameter identifies an invalid device.
PARAM ERR !!UKEYIN	The oplb parameter was missing or contained more than two characters.
WEOF Command	
EOT oplb, device !!UKEYIN	The end-of-tape was encountered.
IL RAD SEQ oplb, device !!UKEYIN	The oplb parameter was assigned to a random-access RAD file.
INV OPLB !!UKEYIN	The oplb parameter identifies an invalid device.
PARAM ERR !!UKEYIN	The oplb parameter is missing.

Table 22. Control Function Command Error Messages (cont.)

Message	Meaning
PRESTORE Command	
CORE OVFLO	Available core memory has overflowed. The Utility program aborts.
PRE ERR !!UKEYIN	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.
ASSIGN Command	
ERR FRGD !!UKEYIN	An attempt has been made to assign a background operational label to a foreground operational label, device-file number, or RAD file.
ERR OPLB1 !!UKEYIN	The operational label to be assigned is invalid.
ERR OPLB2 !!UKEYIN	An attempt has been made to assign one operational label to an invalid or undefined operational label or RAD file.
NO SPARES !!UKEYIN	An attempt has been made to define a new background operational label but no room is available in the corresponding table.
ERR AREA !!UKEYIN	An invalid RAD area name has been used.
OPLB TABLE OVFL !!UKEYIN	An attempt has been made to define more than eight unique operational labels. The assign will be successful, but the operational label will not be used as an output device.

Table 23. COPY Error Messages

Message	Meaning
CORE OVFLO	The memory area used for storing input records (when the CORE option on the !UTILITY COPY command is used) has overflowed. The Utility program aborts.
IL RAD SEQ oplb, device !!UKEYIN	An attempt has been made to copy or verify from or to a random-access RAD file.
OPLB TABLE OVFL !!UKEYIN	An attempt has been made to input more than eight operational labels. Only the first eight unique labels on an !*OPLB card will be entered in the operational label table.
{DEOF oplb, device EOT oplb, device !!UKEYIN}	An end-of-tape, or two consecutive tape marks or !EODs were detected on X4 or UI before the number of files requested has been compared.
EOF oplb, device !!UKEYIN	An !EOD or file mark was detected on X4 or UI before the number of records requested had been compared.
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 aborted.

Table 24. Object Module Editor Error Messages

Message	Meaning
BLNK NAME oplb,device !!UKEYIN	A blank name was input.
CKSM ERR oplb,device !!UKEYIN	A checksum error was detected on a record read from UI or BI.
EOT oplb,device !!UKEYIN	An end-of-tape was encountered on BI or UI.
ILLEG BIN oplb,device !!UKEYIN	The first byte of a record read from UI or BI did not contain X'FF' or X'9F'.
NO name oplb,device !!UKEYIN	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 !!UKEYIN	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.
SEQ ERR oplb,device !!UKEYIN	A sequence error was detected in a record read from UI or BI.

Table 25. Record Editor Error Messages

Message	Meaning
!!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.
LD INPUT UI,device !!UKEYIN	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.
INV CTRL !!UKEYIN	A !*MODIFY control command was interpreted from SI when the Record Editor was not in the modify mode.

Table 26. Sequence Editor Error Messages

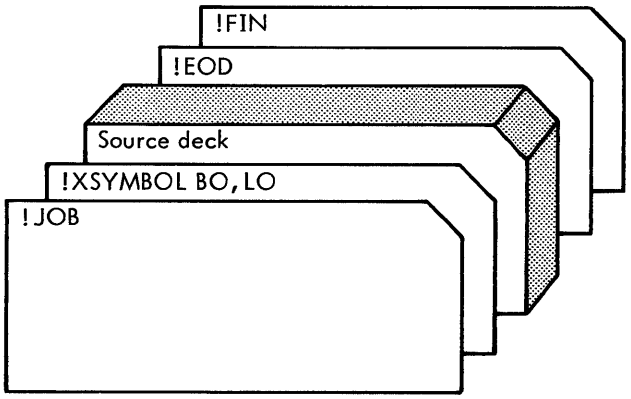
Message	Meaning
DELETE ERR !!UKEYIN	No UI card images were found in the block to be deleted (for !*DELETE and !*SUPPRESS commands).
DEOF UI,device !!UKEYIN	The program to be updated was not encountered on the input tape before the logical end-of-tape. An S response causes the Sequence Editor to return to RBM. All updating done prior to this point has been written, along with the logical end-of-tape marker on the output tape.
PARAM ERR !!UKEYIN	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. 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.
SEQ ERR oplb,device !!UKEYIN	A sequence error was found in either the update data or input tape. In this case, the oplb parameter refers to either SI or UI. An error alarm is also listed on LO.
UNRECOV I/O UI,device !!UKEYIN	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 !!UKEYIN	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.

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.

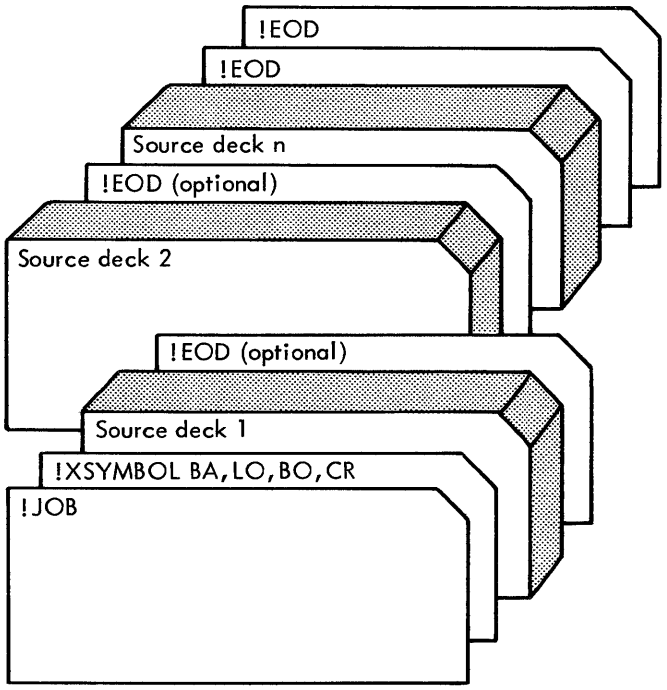
EXTENDED SYMBOL EXAMPLES

ASSEMBLE SOURCE PROGRAM, LISTING OUTPUT AND BINARY OUTPUT



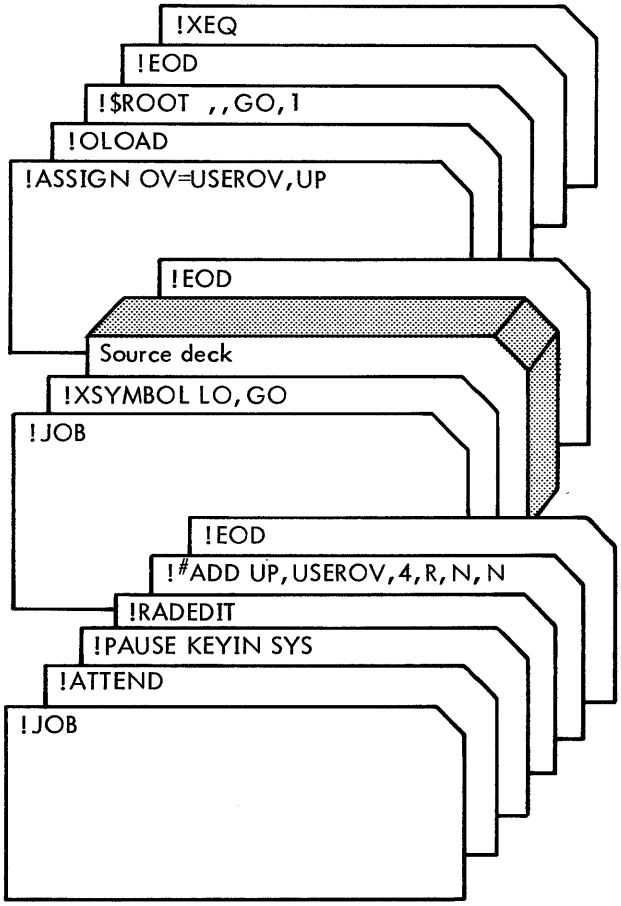
In this example, the symbolic input is received from the SI device (always defaulted), the binary output is received on the BO device, and the listed output is received on the LO device. Note that although BO and LO are normally default cases, they must be specified if output to the GO file (also a default) is not desired.

ASSEMBLE IN BATCH MODE, LISTING OUTPUT AND BINARY OUTPUT WITH SYMBOL CROSS-REFERENCE



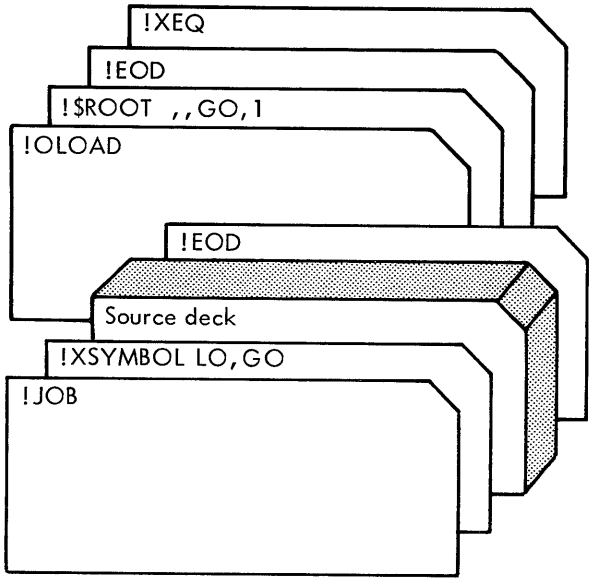
In this example, the source decks are assembled in batch mode (BA). In this mode, successive assemblies may be performed with a single !XSYMBOL command until a double !EOD command is encountered. The parameters defined on the !XSYMBOL command will hold true for each assembly in the batch. Each assembly 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.

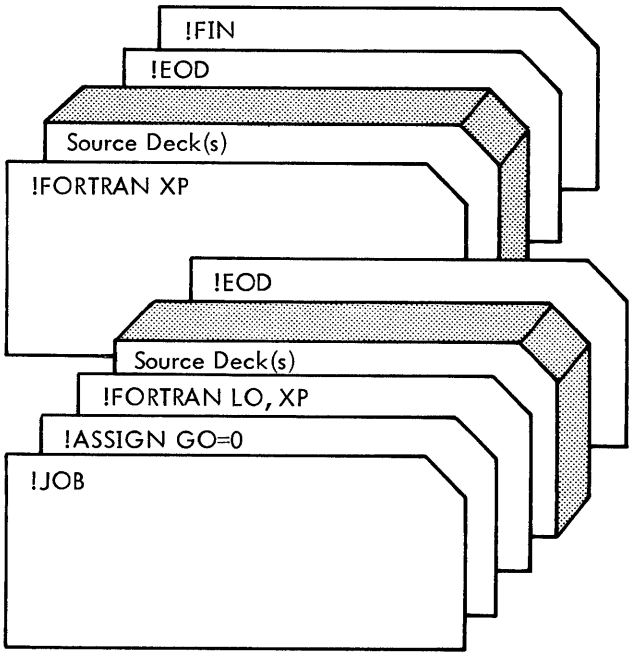
ASSEMBLE SOURCE PROGRAM, LISTING OUTPUT, LOAD AND GO



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. The "1" following the GO oplb specifies that one object module is to be loaded.

BASIC FORTRAN IV EXAMPLES

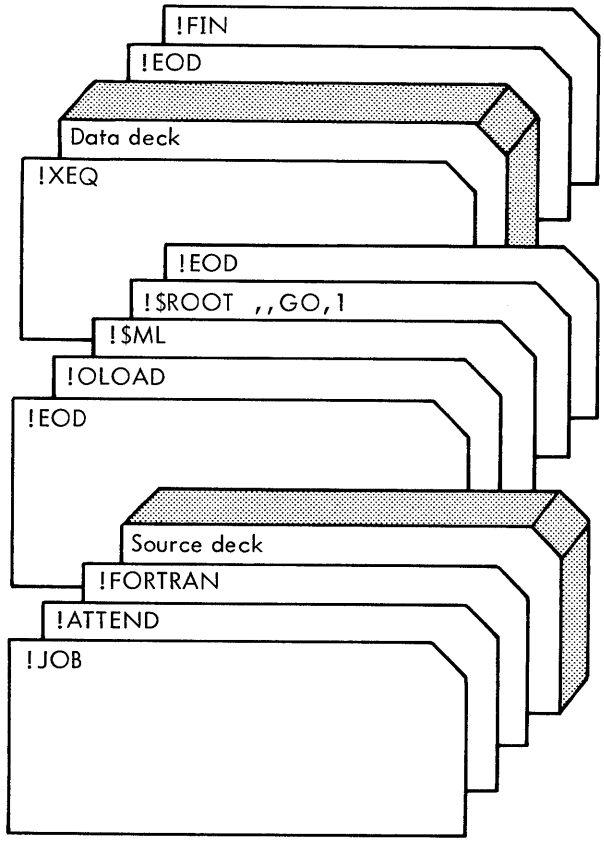
COMPILE MULTIPLE PROGRAMS



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



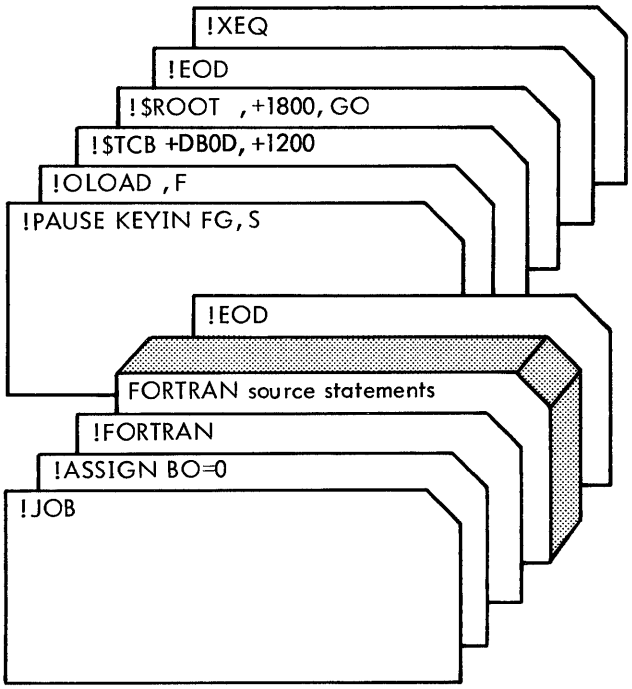
In this example, the !ATTEND command specifies that the Monitor is to go into a "wait" state instead of aborting the job in case of irrecoverable error (generally recommended for "load and go" jobs). Binary output will be received on both the BO and GO devices by default, and standard precision mode 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.

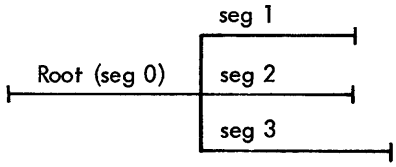


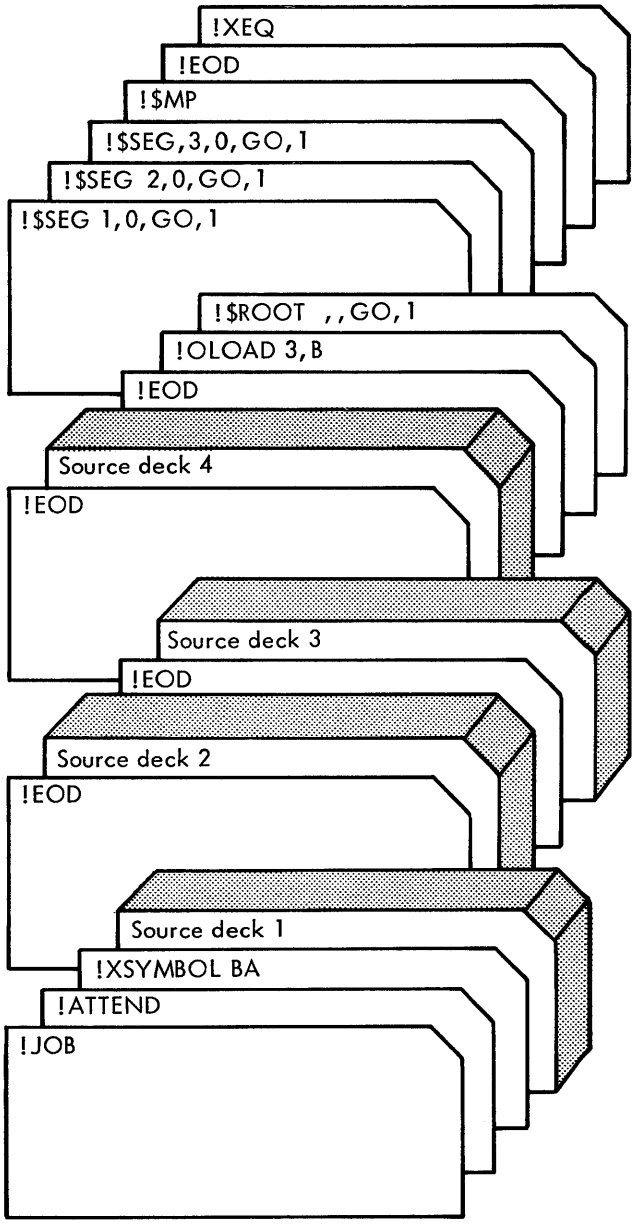
In this example, binary output to the BO device is suppressed. The !FORTRAN control command specifies that the binary output is to be received on the GO file by default and standard precision mode is assumed. The !PAUSE command permits the operator to key in FG, S to access protected foreground memory. 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 IOD (Integral 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

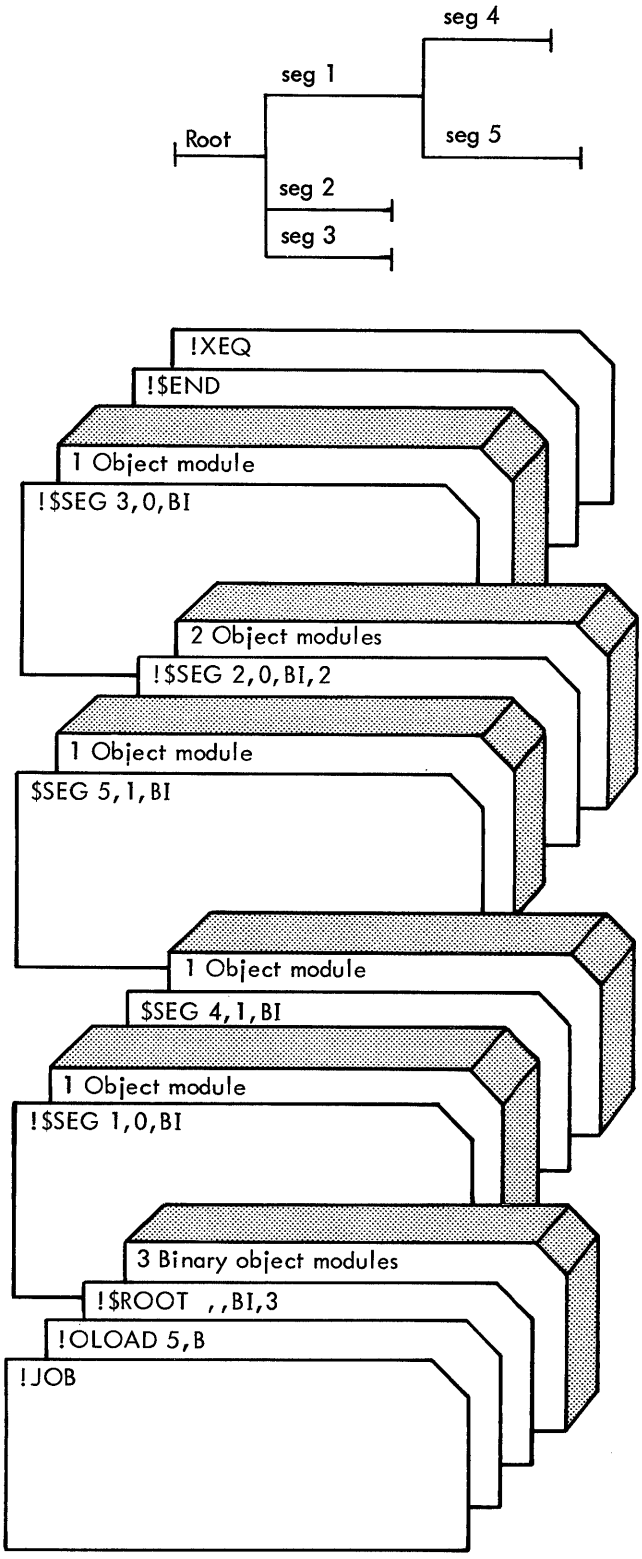




Given the program tree structure shown above, the sample deck setup illustrates a background program with a root and three overlay segments. These are assembled and loaded into the RBMGO file. The !OLOAD command specifies that these three segments are to be loaded, and defines it

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).

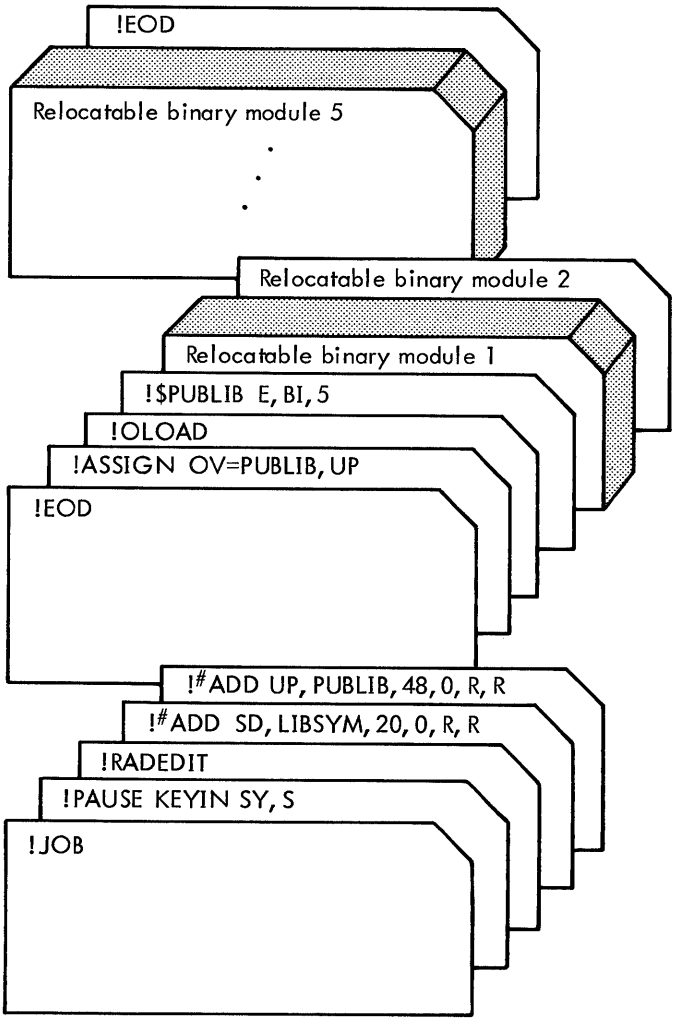
LOAD AND EXECUTE MULTIPLE OBJECT MODULES



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.

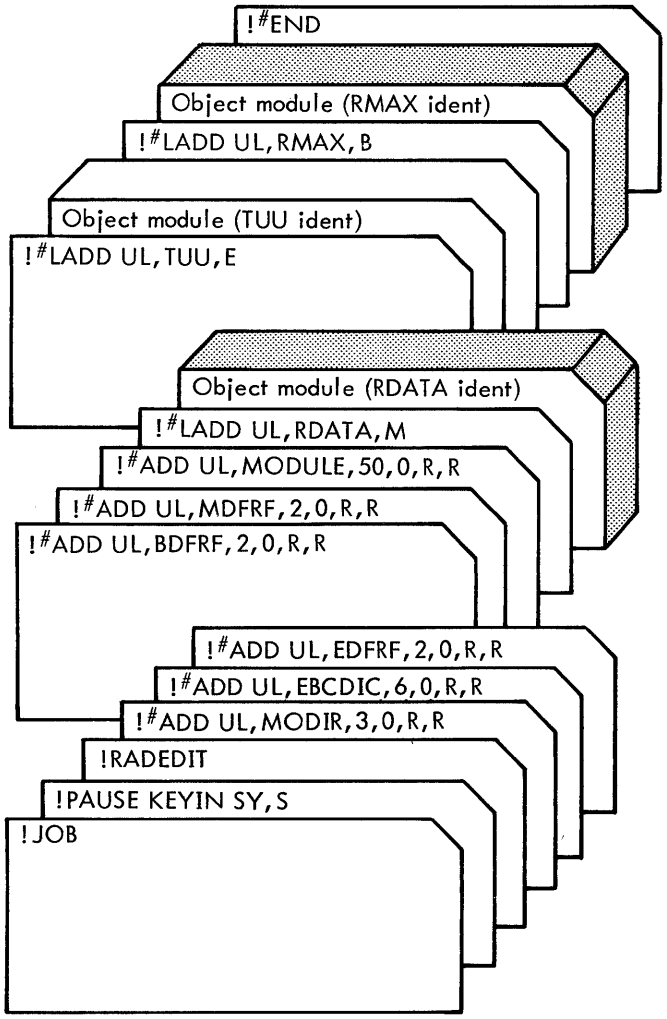
RAD EDITOR EXAMPLES

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.)

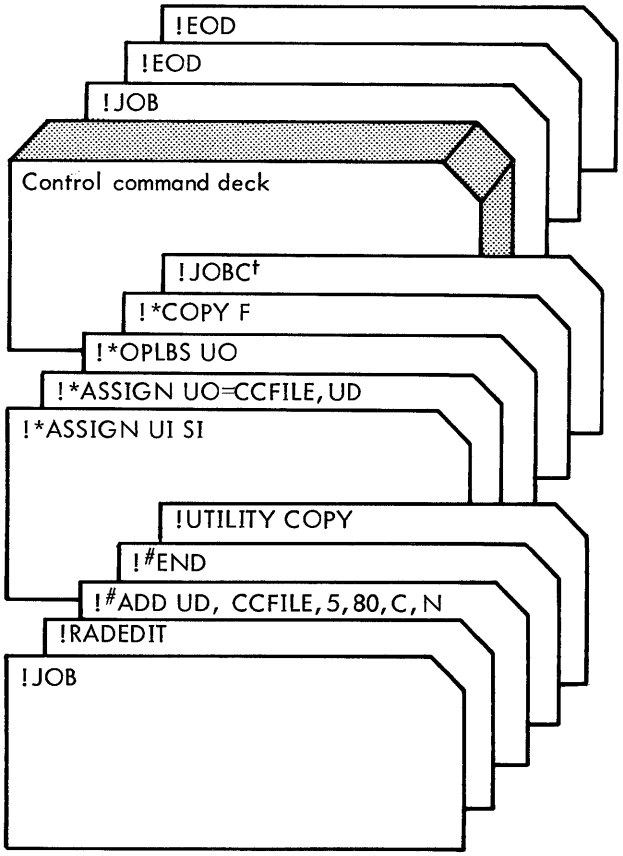
LOAD ROUTINES IN USER LIBRARY



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 GENERATION AND SYSTEM LOAD

INTRODUCTION

An RBM system designed for the requirements of a specific installation is generated in two phases: SYSGEN (System Generation) and SYSLOAD (System Load). These two phases create the Monitor and its required overlays. The SYSGEN phase defines RAD allocation or allows the user to override the nominal area allocation.

SYSGEN loads only the specific installation parameters; none of the processors are loaded at this time. Its only output is an optional, rebootable version of the Monitor. This rebootable Monitor is output on the PM (Punch Monitor) assigned device.

When SYSGEN is completed, core memory is set up for the SYSLOAD function to load the RBM overlays. System processors, user processors, and other user-determined programs are loaded onto the RAD by the Overlay Loader or the RBM Absolute Loader.

It is possible to modify the Monitor and/or its associated processors individually without going through the entire system generation process. Specifically,

- A new version of the RBM can be written without affecting the remainder of the RAD. Therefore, reloading the entire RAD will not be necessary.
- Anything on the RAD can be replaced without going through a SYSGEN as long as the replacements do not exceed their SYSGEN-defined areas.
- One installation can perform a SYSGEN for another installation and merely forward a copy of the rebootable RBM binary deck. However, the recipient facility will have to perform the SYSLOAD; that is, it will have to load the RBM overlays, the system processors, the user processors, and other installation specific programs on the RAD.

SYSGEN

INITIAL CORE ALLOCATION

The RBM system is assembled in two parts. Part 1 is assembled in absolute and contains SYSGEN (and SYSLOAD), and Part 2 is a stack of relocatable binary decks that may be loaded onto the RAD by SYSLOAD. (A list of these modules and their corresponding idents is given in Table 20.) Part 1 is loaded by an Absolute Loader (see !ABS control command in Chapter 2). Nonoptional resident portions of RBM are loaded into the low core (0K-4K) locations from which they will execute; optional resident routines and the system generation routines are loaded into high core (4K-12K). RBM overlays are loaded at SYSLOAD time. The absolute binary deck that includes all optional routines is initially loaded by the Absolute Loader.

After this deck is loaded, the Absolute Loader enters the "wait" state. At this point the operator must enter the device number of the keyboard/printer into the data switches. (The device number used is that of the keyboard/printer employed by SYSGEN to communicate with the operator.) Then the operator may clear the "wait", and SYSGEN will continue.

MINIMUM CONFIGURATION

The following minimum configuration is required for the RBM system generation:

1. Keyboard/printer.
2. Minimum of 16K of core storage.
3. RAD of at least .75M bytes or disk pack.
4. Protection and memory parity features.
5. Hardware interrupts for the RBM Control Task and I/O.

OPTIONAL ROUTINES

There are two basic divisions of the optional routines: those actually resident at all times and those functioning in the overlay region. All of the routines listed in Table 29 function in the overlay region and therefore contribute essentially nothing to the resident size of RBM. The optional resident routines that contribute to the size of RBM are as follows:

Routine	Approximate Size (decimal)
Power On/Off	196
Accounting (Clock 1)	216
High-Speed Line Printer Handler	79
Magnetic Tape Handler	208
Multiply/Divide Simulation	175
M:IOEX	188

The presence of these optional routines is primarily dependent on the installation hardware configuration, which is partly determined as the device-file information is input. If the indicated hardware is present, SYSGEN moves the optional routines to the resident portion of RBM or set the appropriate overlay ident into the overlay table.

For example, if a Y response is given for the INC. MUL/DIV.SIM. query, SYSGEN moves the multiply/divide simulation package that is included in Part 1 to the proper location in core. As another example, if CR4/XX,B is typed under the heading DEVICE FILE INFO, SYSGEN enters the ident of the card reader error recovery routine in the OV:LOAD table. SYSLOAD encounters this ident while loading Part 2, singles out the corresponding module, and saves it as an overlay on the RAD.

Debug and the Character-Oriented Communications handler operate in the foreground; either a resident foreground region or a nonresident foreground area must be allocated if they are to be included.

A method for determining the size of RBM before a SYSGEN is perfomred is given in Appendix I.

CORE MEMORY ALLOCATION

Core memory is allocated in the following manner (see Figure 10):

1. The first 256 words in lower memory (the zero table) are reserved for a communication region (see Table 1).

2. The region from (decimal) 256 to 399 is reserved for internal and external interrupt levels; any space not required for interrupt levels will be used by the Monitor for table space.

3. The remainder of core will be allocated by SYSGEN as follows:

   a. Resident RBM, to be loaded beginning at location 400 (decimal) and to include only optional routines selected by SYSGEN.

   b. Public Library (if allocated).

   c. Resident Foreground (if allocated).

   d. Nonresident Foreground (if allocated).

   e. Background, at least one page whether or not required; minimum amount allocated should be length of the Job Control Processor (3500 locations, decimal). See Figure 11.

4. No foreground space need be allocated for a batch-only system.

When all user inputs necessary to calculate the exact size of the resident RBM are made, the ending address of RBM will be output by SYSGEN. The user will then input starting addresses for the Public Library, the resident foreground, the nonresident foreground, and the background. The user should decide which of these areas are more apt to need additional core space and make the core allocation accordingly. A given area could then expand in a future SYSGEN, but only the programs in that area would have to be reloaded and not the entire system. (In Figure 13, for example, the resident foreground might expand into the unused Public Library area.)

Figures 12 and 13 illustrate the core layout both after absolute load and after SYSGEN and SYSLOAD.

RAD ALLOCATION

During SYSGEN, the total RAD space is divided into a minimum of 1 and a maximum of 20 different areas. Each area is labeled with an area mnemonic, usually from the following list:

SP	UP	CP	Dn
SD	UD	BT	Xn
SL	UL		

where n is a hexadecimal digit.

Areas are allocated by tracks, so the actual size of an area is dependent on the type of RAD device. The various track sizes are given below:

7202/4	2880 words
7232	6144
7242	3072

If the first area allocated to each RAD is not preceded by an SK (skip track) input, the system bootstrap will be written in sector 0, and the area will actually begin at sector 1. All other areas, with the possible exception of the BT area, will always start on a track boundary. The five areas described below may receive default allocations. During RAD allocation, the user must specify a system RAD to receive the default areas. An SK input as the last input on the system RAD will be ignored if default allocations are to be made.

The areas that may be default allocated and their sizes are

- SP — Only large enough to contain RBM overlays and all standard processors (see Table 6). This is the only mandatory area.
- SD — Only large enough to contain nominally large RBM GO and RBM OV files and other small files (i.e., RBM S2, RBM ID etc.).
- SL — Only large enough to contain the standard system libraries: standard precision, extended precision and common, or main libraries.
- CP — Only large enough to contain all of background.

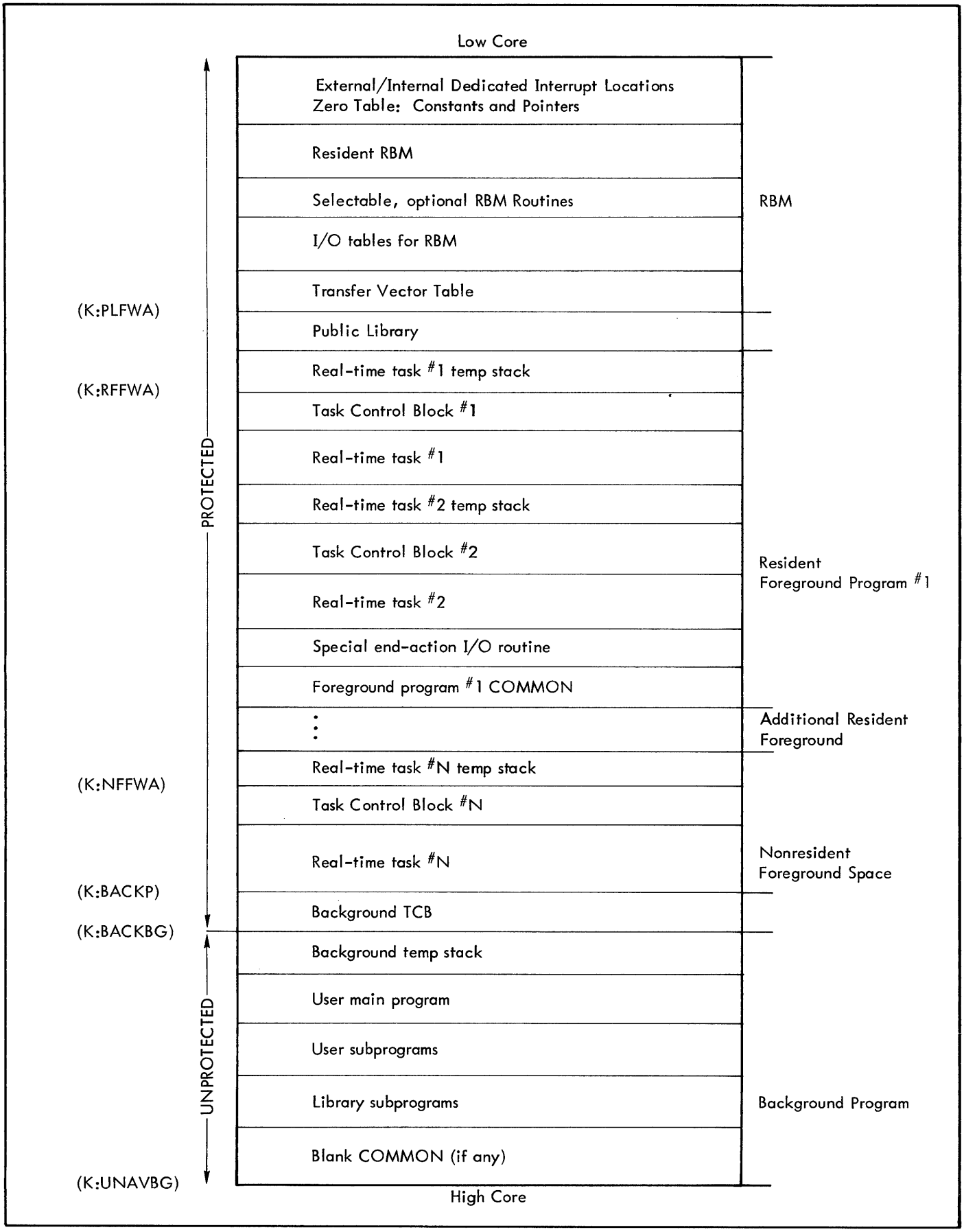


Figure 10. RBM Core Memory Allocation Example

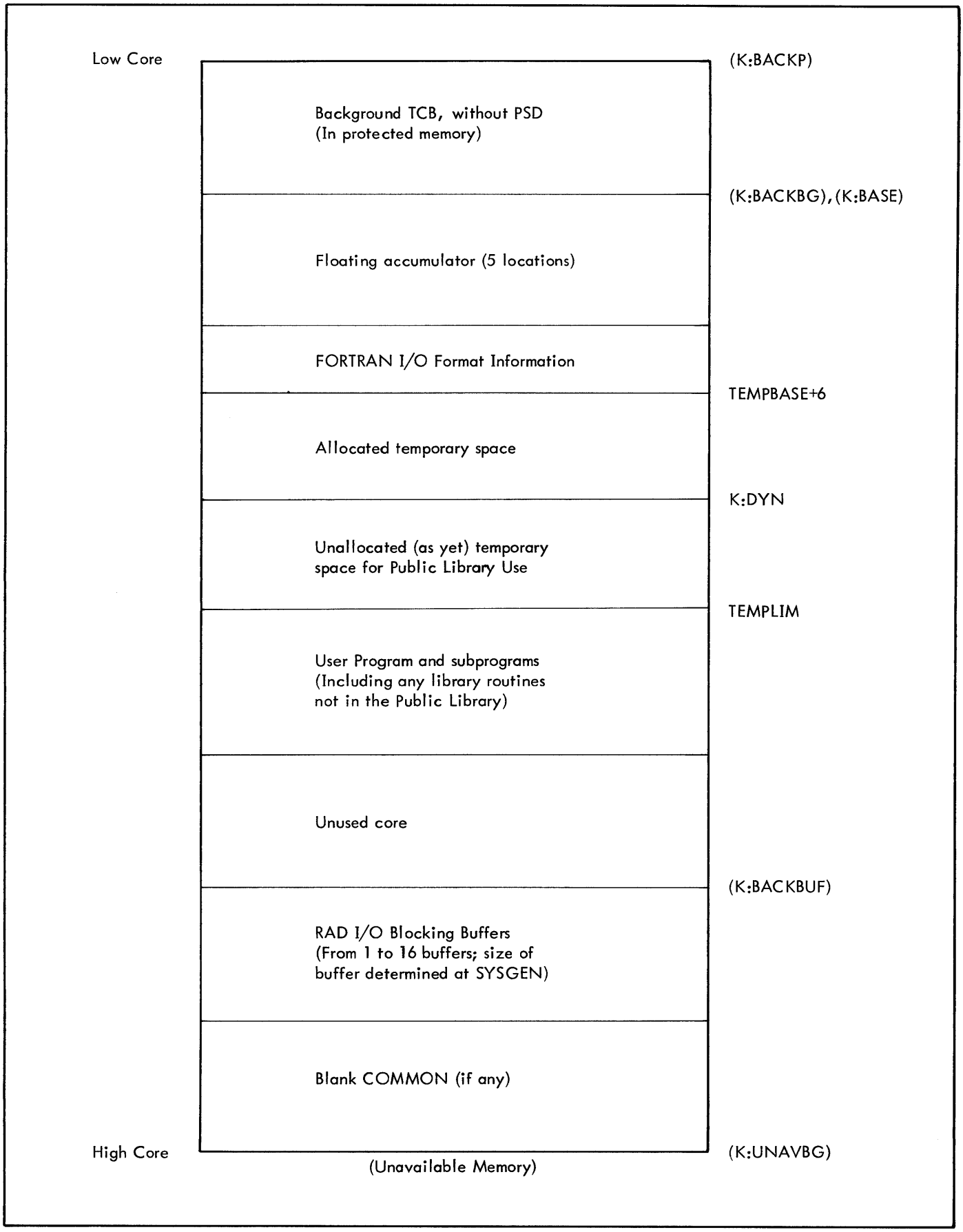


Figure 11. Background Core Allocation Example

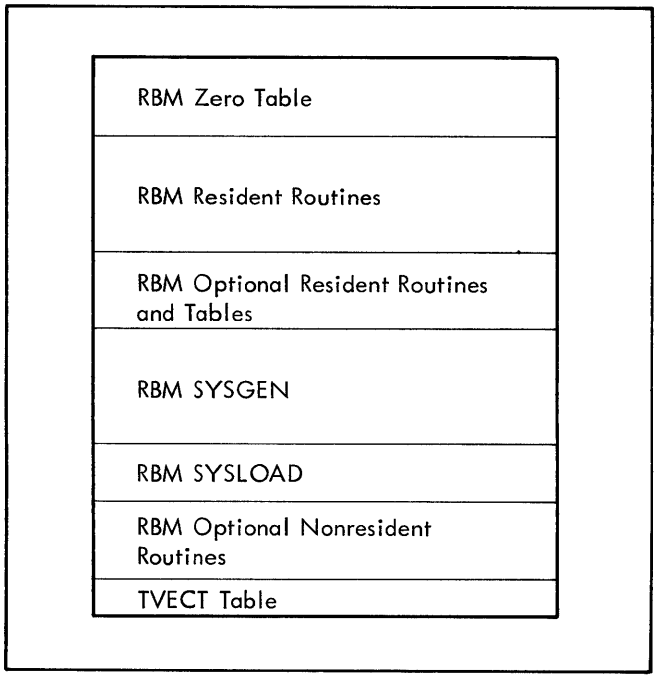


Figure 12. Core Layout After Absolute Load

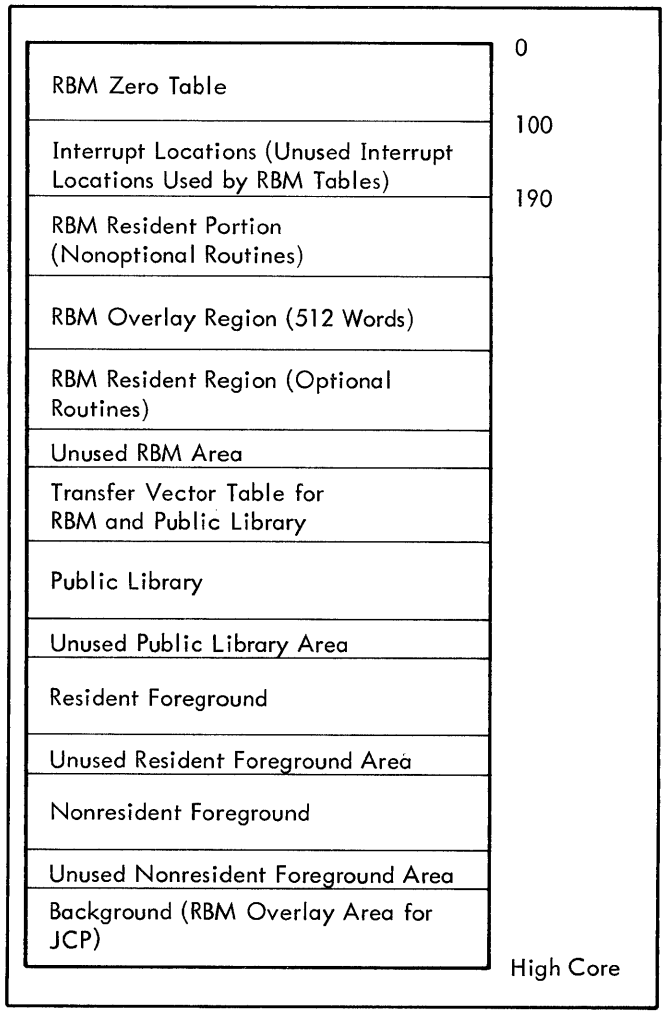


Figure 13. Core Layout After SYSGEN and SYSLOAD

BT Remaining RAD space. The last track available for the default assignment of this area is device specific, as follows:

Device	Last Track Available+1
7202	123
7204	506
7232	506
7242	4000

For all devices except a 7242, all available tracks may be allocated. Tracks at the upper end of the device are used as alternates for bad tracks within an area. The RAD allocation at SYSGEN constructs the Master Dictionary, consisting of four words per entry. The only restrictions are that each area mnemonic must be alphanumeric, the size of the Master Dictionary may not be exceeded, and if an area is allocated twice, the space originally reserved will be lost.

FILE CONTROL TABLE ALLOCATION

The File Control Table (FCT) is indexed by device-file number and contains information about all device-files in the system. The total size of the File Control Table is determined and allocated at SYSGEN time. The term "device file number" (DFN) indicates the order in which devices are defined. For example, since the first device defined must always be a keyboard printer; DFN 1 will always specify a keyboard printer. Devices other than the RAD have permanent device-file number assignments made at SYSGEN time. SYSGEN allows room for up to 50 permanent device-files (not including RAD files).

A separate device-file (i.e., FCT entry) is required for each open file on the RAD. Hence, the total number of entries necessary in the File Control Table for all RAD files is the maximum number of simultaneous open files. At SYSGEN time, the user must specify this maximum number of device-files for his foreground programs. For the background, nine device-files will be allocated (a sufficient number for the system processors), if the user does not choose to override the default case.

SYSGEN always allocates three foreground RAD files for use by the Monitor in addition to the number of RAD foreground files input by the user. Hence, the total size of the File Control Table will be the sum of the number of non-RAD files assigned, plus the total number of RAD files reserved for foreground use plus three, plus the number of RAD files reserved for background use (nine, if none are explicitly reserved).

The user can make file dictionary entries on the RAD for his foreground programs and then permanently allocate a foreground device-file number to that RAD file by assigning

the RAD file to a foreground operational label. A device-file number reserved for background use is assigned by the Monitor service routines M:DEFINE and M:ASSIGN whenever a call is made to either of these routines. For RAD device-files, SYSGEN allocates the appropriate space in the File Control Table and sets the background/foreground indicator, the "file for RAD use" indicator, the maximum retry counter, and the pointer to the I/O Control Table. For non-RAD files, SYSGEN sets in the File Control Table the background/foreground indicator, the channel number, the device type number, the "file for non-RAD use" indicator, the device number, the maximum retry counter, and the pointer to the I/O Control Table.

SYSGEN also allocates space for the I/O Control Table. The amount of space required for each type of device is contained in the Device Type Table.

OPERATIONAL LABEL ASSIGNMENTS

During SYSGEN the user specifies the selected standard operational labels and assigns each to a device-file number (other than a RAD file number) or to device-file zero. These assignments will be maintained as permanent assignments for the appropriate operational label.

The operational labels listed below are normally associated with RAD files. Therefore, permanently assigning these labels to non-RAD files at SYSGEN time is not permitted.

Operational Label	Use
RM	Used by RBM to load the RBM overlays and is reserved exclusively for RBM.
ML	Used by M:LOAD to load nonresident foreground programs.
PI	Should be used by any background program with overlays to load the overlay segments from the RAD. For system processors, PI is assigned to the processor file. For background programs loaded with an XEQ command, PI is assigned to OV. Foreground programs must specifically assign an operational label to the file from which overlay segments are to be read.
OV	Normally assigned to the RBMOV file for "assemble and go" type operations.
X1-X5	Processor scratch files.
S2	XSYMBOL standard procedures.
GO	Normally assigned to the RBMGO file for "assemble and go" type operations.

After all inputs are made by the user, SYSGEN allocates three additional entries in the Foreground Operational Label Table for RAD foreground labels.

A total of 100 operational labels can be allocated and assigned at SYSGEN time, including those automatically allocated by SYSGEN.

INPUT PARAMETERS

When RBM is loaded and control is transferred to the SYSGEN routine, operator intervention is required to input the system parameters. The following device types are standard and must be referred to by name when inputting the device-file definitions:

SYSGEN Device Type Name	Device Characteristics	Device Name
KP	Keyboard/printer	KP
M9	Magnetic tape, 9-track	MT
PT	Paper tape handler	PT
M7	Magnetic tape, 7-track, packed binary option	MT
B7	Magnetic tape, 7-track BCD option	MT
RD[†]	RAD or disk pack	RD
XX	Special-purpose device for use with M:IOEX	-
LP2	Line printer, 240 lpm	LP
LP8	Line printer, 800 lpm	LP
CR4	Card reader, EBCDIC option, 1400 or 400 cpm	CR
BR4	Card reader, BCD option, 1400 or 400 cpm	CR
CP3	Card punch, 200 cpm	CP
BP3	Card punch, BCD option, 200 cpm	CP
CP1	Card punch, EBCDIC option, 100 cpm	CP

[†]RD is used only to reserve a specific number of foreground or background RAD files, not as a name of the form dtnn.

SYSGEN Device Type Name	Device Characteristics	Device Name
BP1	Card punch, 100 cpm	CP
PL	Graphic plotter[†]	PL

[†]RBM supports the graphic plotter as a device type but will not do any special converting or formatting. The user can either use the existing library routines to format data for the plotter or perform his own formatting.

The Run-Time names are used by M:READ/M:WRITE for operator communication.

Table 27 defines the system parameters that are input via the keyboard/printer, paper tape reader, or card reader during SYSGEN. Note that all numeric entries can be input in either decimal or hexadecimal with leading zeros ignored; all hexadecimal entries must be preceded by a +. Comments can be added to any input by leaving one space after the required input is made. All inputs from the keyboard/printer must terminate with a NEW LINE code. Commas are used to separate fields. If an input/output device is not in the START state, an appropriate message will be written on the keyboard/printer.

Table 27. SYSGEN Input Options and Parameters

Output Message	Input Parameters	Description
!!RBM SYSGEN INPUT DEVICES	Device Name and Number (e.g., CR4/03, LP8/02;KP, NO;PT20, KP)	Device name and device number of the input and output devices to be used during SYSGEN. If the keyboard/printer is to be used exclusively, only KP need be input. The only acceptable device names are CR, LP, KP, PT, or NO.
VERSION	Two alphanumeric characters (e.g., A1 or A2 or B1, etc.)	The RBM version will be stored in a zero table location, K:VRSION, output by RBM on LL at the start of each job and by postmortem dump whenever it runs.
MEMORY SIZE	Numeric size	Total core memory size of Sigma 2/3, stored in a zero table location, K:UNAVBG.
MAX. INT. LOC.	Address	Maximum Sigma 2/3 address for real-time external interrupts ($263 < A < 400$).[†] The space unused by the interrupts will be allocated to RBM tables by SYSGEN.
CONTROL TASK INT. LOC.	Address	Address of interrupt used by RBM Control Task. Must be the interrupt with the lowest priority available.
INT. CHANNELS EXT. CHANNELS	x - y or 0	Indicates the numbers of the I/O channels specific to this installation. x is the first channel number, and y is the last number. If no channel exists for this IOP, a 0 is input. Sigma 2, for example, would always have an input of 0 for EXT. CHANNELS. The number of channels must be greater than four but less than 20 for Sigma 2 (less than 28 for Sigma 3). For Sigma 3, 0 through X'B' are the internal channel numbers and X'C' through X'1B' are the external channel numbers.

[†]Although Sigma 3 has provisions for interrupt locations only as high as 368, 400 is considered to be the beginning of operating RBM for compatibility with Sigma 2. The 32 extra cells are used for input/output tables.

Table 27. SYSGEN Input Options and Parameters (cont.)

Output Message	Input Parameters	Description
NO. LINES/PAGE	Number	Number of lines to be printed on each page during an Extended Symbol assembly. SYSGEN will save the input value in zero table location K:PAGE, for later use by Extended Symbol in printing out a title at the top of each page. Input value n must be $0 < n < +8000$.
NO. DEFS IN PUB. LIB.	Number	Number ($n < +100$) of definitions (DEFs) in the Public Library. This input is needed so that the Transfer Vector Table can be correctly allocated. If zero is input, SYSGEN assumes there is no Public Library.
NO. ENTRIES IN NONRES. FGD. QUEUE	Number	Reflects the maximjm queue size for nonresident foreground programs.
NO. FGD PARITY ERRS	Number	Number of parity errors to allow in foreground before disabling the foreground task.
NO. DICT. ENTRIES	Number	Specifies the length of the Master Dictionary. Entries are already allocated for SP, SD, SL, CP, and BT. A number from 0 to 15 may be input, specifying the additional Master Dictionary entries. Each entry requires four words.
ALT. TRK. POOL SIZE	Number	Specifies the length of the Alternate Track Pool, which will contain bad track numbers. It should be at least as large as the maximum number of known bad tracks. The bounds are 0 to 512.
RAD ALLOCATION	RDxx/dn , {I / E} ,S (for example, RD42/E1, S)	xx specifies the device type as follows: 02 7202 03 7203 04 7204 32 7232 42 7242 dn is the hardware device number for this RAD, which must be driven by a channel defined previously under "INT CHANNEL" or "EXT CHANNELS". Each device can only be input once, but as many as 12 devices, each with area allocations, may be input. I or E specifies the IOP type; I refers to an Internal IOP, and E to an External IOP. E is assumed for a 7242 or 7246 and is the default case for a 7232. I or E must be input for a 720x. If this parameter is not used, an intervening comma before the next parameter is not necessary. S indicates that this device is to receive default allocations. If more than one S parameter is input the last is used. If S is not input, the device receiving the SP area is used. Either S or the SP area must be input.

Table 27. SYSGEN Input Options and Parameters (cont.)

Output Message	Input Parameters	Description
RAD ALLOCATION (cont.)	yy = zz For example: SP = 30 SD = 20 D1 = 100 D2 = 200	yy is any area mnemonic, usually from the following list SP UP BT Dn SD UD CP Xn SL UL where n is a hexadecimal digit. zz is the number of tracks to allocate for area yy. If zz = 0, area yy will be undefined, and an additional Master Dictionary entry will be available. If zz = ALL, the area will occupy the remainder of the RAD and no other inputs may be made for this RAD. If yy = SK, zz number of tracks will be skipped before the next area is allocated. But to be meaningful, another area must be input. If the first input is not SK = zz, this RAD will receive a system bootstrap in sector 0 and the next area will actually begin in sector 1. If no orders are allocated on RAD dn, however, no bootstrap will be written.
	END	Terminates the RAD ALLOCATION parameter. In the example given, areas SP, SD, D1 and D2 will receive the number of tracks specified. SL, CP, and BT will be default allocated, (as described under RAD ALLOCATION) on this same device.
BUFFER SIZE	180 or 512	Specifies the blocking buffer size for all Monitor blocked files in this system.
INC. POWER ON/OFF	Y or N	Yes (Y), if Power On/Power Off routine is to be included in resident RBM.
INC. MUL/DIV. SIM.	Y or N	Yes (Y), if multiply/divide software is to be included. If multiply/divide hardware exists, No (N) should be input.
INC. M:IOEX	Y or N	Yes (Y), if optional RBM service routine M:IOEX is to be included.
INC. CLOCK ONE	Y or N	Yes (Y), if Clock 1 is to be used by RBM for job accounting, for limiting the execution time of background jobs, for time limits on I/O transfers, and for keeping time of day. If No (N) is input, Clock 1 is not available and SYSGEN will not load the job accounting portion of the RBM Control Task.
INC. DEBUG	Y or N	Yes (Y), if RBM Debug is to be included. If Debug is included, at least $200_{(16)}$ foreground cells must be allocated and Debug I/O devices may be input below, under DEVICE FILE INFO. If No (N) is input, Debug will not be loaded and the user can use the 32 zero table Debug cells as additional foreground mailboxes.

Table 27. SYSGEN Input Options and Parameters (cont.)

Output Message	Input Parameters	Description
INC. MISC.	Y or N	Yes (Y), if the non-Debug Core Dump, RAD Dump, and Hex Corrector routines are to be included in RBM. If a Y response is given, the resident size of RBM will increase by $85_{(10)}$ cells.
INC. C.O.C.	Y or N	Yes (Y), if Character-Oriented Communications Handler is to be included. If COC is included, at least 1000 cells must be allocated for resident foreground.
DEVICE FILE INFO. [(INC. DEBUG)]	dtnn,x ,[{I / E}]	The first parameter, dt, specifies a certain peripheral and must be one of the device type names listed previously under "Input Parameters". The second parameter, nn, is the hardware device number of this peripheral and must indicate a previously defined channel. The third parameter, x, is F if this is a foreground device, B if this is a background device, DI if this is a Debug input device, or DO if this is a Debug output device. (DI and DO will not be accepted if an N (no) response was given to the INC. DEBUG message.) The last parameter, I or E, is required to indicate IOP type for a multiunit device; I indicates an internal IOP, and E indicates an external IOP. The last parameter is ignored if the device is not a multiunit type. The first device-file entry, DFN 1, must be KPnn,F. The term "device-file number", abbreviated as DFN, indicates the order in which device parameters are input in response to the DEVICE FILE INFO. output message.
	RD, x, y	This entry indicates to SYSGEN that y RAD File Control Table entries are to be saved for the mode specified by the parameter x (same as x above, except that DI and DO cannot be used). The y parameter may be one or two decimal digits. An entry must always be input for the foreground, and a default number of 9 is used for background files if a user fails to allocate any background file. (Thus if no input is given, SYSGEN will reserve nine background RAD file entries.) The value 9 is always added to the background allocation. Examples: DFN No. — Device-File 1 — KP40,F 2 — LP8/02,B 3 — CR4/03,B 4 — CP1/04,B 5 — PT20,B 6 — BR4/03,B 7 — M9D0,B,E 8 — M9D1,B,E 9 — M7E0,B,E

Table 27. SYSGEN Input Options and Parameters (cont.)

Output Message	Input Parameters	Description
DEVICE FILE INFO. [(INC. DEBUG)] (cont.)	RD,x,y (cont.)	Examples: DFN No. — Device-File 10 — B7E0,B,E 11 — LP2/05,F 12 — CR4/03,F 13 — M9D0,F,E 14 — M9D1,F,E 15 — XXD0,F 16 — LP8/02,D0 17 — CR4/03,D1 18-27 — RD,B,10 28-46 — RD,F,20 END
	END	Signifies end of device-file information.
BCKG. OP. LBL.	Operational label = device-file number, or device unit number=device-file number (one per line, terminated by END); 0=n means reserve n locations in Operational Label Table for temporary assignments. (Temporary space is needed for execution time temporary assignments, or for RAD files above and beyond that number (9) which is automatically allocated by SYSGEN). Examples: SI=3 102=4 0=3 (reserves three additional entries in Operational Label Table)	Background operational labels or device-unit number and device-file number equivalents for permanent I/O assignments. No operational labels can be assigned to RAD files at SYSGEN time. A maximum of 188 background and foreground operational labels can be input by the user. The following operational labels are defined by RB7; thus, they may not be input: RM OV ML GO PI
	END	Signifies end of background operational label.
FGD. OP. LBL.	Same as for background, except that space for three operational labels is automatically assigned.	Foreground operational labels or device unit number and device-file number equivalents for permanent foreground I/O assignments. No foreground operational labels can be assigned to RAD files at SYSGEN time.
RBM LWA = + xxxx	None	At this point, SYSGEN will have sufficient information to calculate the exact size of RBM. This message is output to the operator as an aid in the follow-on inputs. If the user has only background, he will have to input an address for the start of the background (i.e., at least 38 cells greater than the RBM LWA output). This value (i.e., + xxxx) can be predetermined by using the algorithm given in Appendix L.

Table 27. SYSGEN Input Options and Parameters (cont.)

Output Message	Input Parameters	Description
PUB. LIB. FWA†	Address	If zero has been input for the number of DEFs in the Public Library, this typeout will not occur. Otherwise, the input should reflect the first word address of the Public Library (which may be equal to RBM LWA). An input of zero is illegal. This value is stored in zero table location K:PLFWA.
RES. FGD. FWA†	Address	First word address of the resident foreground area. An input of zero indicates no resident foreground. This value is stored in zero table K:RFFWA.
NONRES. FGD. FWA†	Address	First word address of nonresident foreground area. An input of zero indicates no nonresident foreground. This value is stored in zero table location K:NFFWA.
BCKG. FWA†	Address	First word address of background memory. This address must start on a page boundary (some multiple of 100_{16}). This value is stored in zero table location K:BACKBG.

†These four addresses must be in increasing order. That is, the core allocation must be made in the same order as the SYSGEN input. If nonresident foreground is used, it must be at least 218 cells long. This area is used as a buffer for the 'Q' key-in.

SYSGEN OUTPUT

MESSAGES TO THE OPERATOR

The error messages in Table 28 can be output by SYSGEN. Note that for input errors (except for an allocation error), the corrected input must be made from the KP exclusively.

BINARY OUTPUT

If a background PM (Punch Monitor) operational label is assigned at SYSGEN time, SYSGEN will punch a rebootable version of the RBM on the PM device after the last parameter has been input by the operator.

SYSLOAD

SYSTEM LOAD

After SYSGEN has been completed, or the rebootable RBM deck punched by SYSGEN has been input, control is transferred to the System Load Processor, SYSLOAD. SYSLOAD will initailly output the following message on the OC device:

!!RBM SYSLOAD

!!INPUT OPTION

The option to be input on OC should be either one of the following:

PA specifies that patches are to read from the input device, with the format xxxxbbyyyy[bbzzzz]... !EOD where xxxx is the location to be patched, and yyyy and zzzz are the values to be inserted at location xxxx. All entries must be four characters long, separated by two blanks. The !EOD terminates patching and causes the !!INPUT OPTION message to be output again. The ALL or UPD option can then be entered.

ALL which specifies that a complete system load is to occur and nothing on the RAD is to be saved;

or

UPD which specifies that an updated version of RBM has been made to replace the existing RAD version. Portions of the RAD may have to be reloaded, depending on the new core memory allocation.

ALL OPTION

An ALL input specifies that a complete system load is to occur. A complete load is necessary for the initial generation or whenever any of the RAD areas has to change size.

The System Load Processor (SLP) first searches the Master Dictionary left by SYSGEN to determine if any RAD areas have not been completely defined because of an ALL input during SYSGEN. If some areas still need their last word

Table 28. SYSGEN Error Messages

Message	Meaning	Recovery
!!INVALID PARAMETER	Input parameter is out of expected range, or maximum number of allowable inputs have been made.	Retype input with correct value.
!!FORMAT ERR	Input format not valid.	Retype input with valid format.
!!C. T. INT. PRIORITY ERR	Control task interrupt is at a higher priority level than the I/O interrupt level.	Requires hardware modification, or reassignment of Control Task Interrupt to a lower level.
!!I/O ERR	An I/O error has occurred on the last input.	Correct the problem with the input device and retype last input.
!!ALLOCATION ERR	No RAD was defined as the system RAD.	Since this alarm is output only after the END card is input (i.e., after the RAD allocation has been completed), the user must reallocate all areas assigned to the system RAD. The default allocations will be restored for the second iteration. The computer will enter a "wait" state so that the error can be isolated and corrected unlike other SYSGEN errors. The corrected inputs must be made on the original input device.
TOO MANY AREAS	Not enough entries were defined in the Master Dictionary.	Fewer entries must be input or more Master Dictionary entries must be made available. In any event, RAD allocation must be restored.
RBM CAN'T BOOT	RBM resides on a 7242 disk and crosses a cylinder boundary.	SP must be reallocated during a second RAD allocation.
!!ILLEGAL OP. LBL.	The user has attempted to permanently assign one of the reserved op labels (RM, ML, PI, OV, GO).	Retype input with different op label.

addresses defined, SLP will generate this value so that the Master Dictionary can now be completed.

At this point a check is made to determine if the checkpoint area is large enough to contain the entire background. If it is not, the following message will be output:

CP AREA TOO SMALL

The CP area will be undefined.

This error is only fatal if an attempt is made to checkpoint the background. It can be corrected only by a complete SYSGEN, using at least the default size for the checkpoint area.

After the Master Dictionary has been completed, the SLP will write zeros on all defined areas of the RAD. This process takes approximately 1 minute for a 256-track RAD.

LOADING RBM PART 2

At this point the SLP outputs the following message:

!!LOAD RBM PART 2

If SLP encounters a track upon which it cannot write, an appropriate message will be output and that track number will be entered in the Alternate Track Pool. For a disk device, SLP will clear only the first sector of each area, and will obtain bad track numbers for the Alternate Track Pool from the headers of tracks 4000 to 4360.

SLP will then write the following into the first sector of each area: the area mnemonic, the bounds of the area, and a bad track list for all bad tracks on that device. SLP will also clear the second sector of each area.

The binary modules making up Part 2 should be input from the background AI device (as determined at SYSGEN).

So that a user does not have to reorganize Part 2 for each new SYSGEN, SYSLOAD allows all or Part 2 to be input each time, but only loads the routines specified by the options selected during SYSGEN. The final module must be followed by an !EOD. The ident from the Extended Symbol directive, IDENT, is used to identify each module loaded, and is placed in the OV:LOAD table and used as the overlay identification.

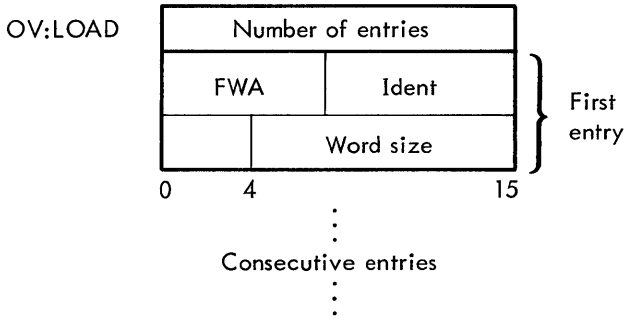
RBM PART 2

The routines making up RBM Part 2 and their idents are listed in Table 29.

Table 29. Routines and Idents for RBM Part 2

Group	Overlay	Ident (hexadecimal)
Monitor Service Routines	M:ASSIGN	A1
	M:DEFINE	A2
	M:OPEN	A3
	M:CLOSE	A4
	M:LOAD	A5
	M:DOW	A6
	M:WAIT	A7
	M:CTRL	A8
	M:RSVP	A9
	M:DATIME	AA
	M:COC	AB
	RAD Bootstrap	AC
Control Task Subtasks	S:CKPT	1
	S:REST	2
	S:LOAD	3
	S:ABORT	4
	S:TERM	5
	S:KEY	7
	S:KEY2	71
	S:KEY3	72
	S:KEY4	73
	S:PMD	8
	S:CCI	B
	S:PARPWR	FF
	Power Failure	B1
Background	BACKCCI	10
Debug	All Overlays	20-2F
Device-Dependent Error Recovery Routines	All	30-3F
Miscellaneous Routines	Core Dump and RAD Dump	40
	Hex Corrector	41
	FCT Dump	42

SYSLOAD loads the required overlays, absolutizes them for their execution location, and writes each overlay on the RAD in an unpacked format. Only one overlay can occupy the overlay area in memory at any one time. SYSLOAD stores the RAD address (as a displacement) and the word count of each overlay in the RBM OV:LOAD table. The OV:LOAD table has the following format:



where FWA is the starting sector number (relative to the beginning of the system processor area) of this overlay. All overlays start on a sector boundary. No overlays cross a track boundary.

If an error condition occurs during the loading of the individual modules making up RBM Part 2, the following message is output:

```
XX ERR, ID:YY

??RETRY?
```

where

XX is one of the following error types:

XX	Error Type
CS	Checksum.
SQ	Sequence.
TY	Item type; no external references or definitions are allowed.
BI	Binary deck is incomplete.
OG	Origin error; an attempt has been made to re-origin a portion of this routine to a region already on the RAD.

YY is the ident of the current routine (if the ident is unknown, YY = ??).

The response to the RETRY query can be either N (no) or Y (yes). If the response is N, the SLP skips to the next routine. If Y is input, the current routine is left as is and an attempt is made to continue with the next card; for some of the above errors, however, continuing in this manner may be undesirable.

After loading all of RBM Part 2, SYSLOAD determines if all required routines are present. If some routines are missing, the following alarm is typed:

!!MISSING IDENTS: xx xx xx xx ...

??RELOAD?

where xx is the ident (Extended Symbol directive IDNT) corresponding to the missing routine.

If Y is input to the reload query, SYSLOAD again reads the AI device to load the missing routines. This sequence is repeated until all required routines are loaded or until an N is input.

After RBM Part 2 has been loaded, entries will then be made in the System Processor Dictionary for RBM, the Transfer Vector Table, and the RBM bootstrap. Each of these items is assigned in a separate file in the system processor area of the RAD.

After the nonresident portion of the RBM is on the RAD, the resident portion is written. SYSLOAD calculates the IOCDs needed to read RBM into core storage and stores the information into the last part of the RAD bootstrap.

After RBM is written on the RAD, the Transfer Vector Table will be written onto the TVECT file. The Transfer Vector Table contains transfer vectors for Monitor service routines and Public Library routines. The amount of RAD space allocated for the TVECT file depends on the maximum number of DEFs in the Public Library, which is a SYSGEN input.

The final program output to the System Processor area of the RAD will be a copy of the RBM bootstrap that goes into the BOOT file. There is no file header for the bootstrap, and the bootstrap is always restricted to one sector. It is necessary to define the bootstrap as a file, so that it can be accessed for output during a RAD save or dump operation. After the bootstrap is written onto the BOOT file, it is written onto relative sector zero of the system RAD, from where it can be bootstrapped into core. Also, a copy of the RAD bootstrap may be output to the foreground BO device, which enables the user to start RBM on any sector of the RAD or to boot from a disk pack. If the user chooses to start RBM at any sector other than sector zero, he can still reboot RBM by loading the RAD bootstrap that was punched on the BO device.

The next output to the RAD will be the RBM Symbol Table (a file in the System Data area) and the System Data Area Dictionary. The System Data Area Dictionary has the same format as the System Processor Dictionary and contains the following files:

File Name	Description
RBMGO	Object module storage for "assemble and go" operations.
RBMOV	Nonpermanent storage for programs to be executed.
RBMS2	Storage for Extended Symbol standard procedures.
RBMSYM	RBM Symbol Table of Monitor service routines.
RBMPMD	RAD area used by postmortem dump.
RBMID	Holds IDNT origins for Debug.
RBMAL	Used by the accounting routine.

If a user accepts the default allocation for the RBMGO, RBMOV, RBMAL, RBMID, RBMS2, and RBMPMD files, no modifications via the RAD Editor have to be made for these files.

The RBM Symbol Table contains the definitions (DEFs) for the Monitor service routines. These DEFs are needed by the Overlay Loader at load time to satisfy any reference to the Monitor service routines. The first word of the table contains the number of bytes in the table, followed by seven words per entry, in the same format as in the Overlay Loader Symbol Table.

After the System Load Processor completes its writing of the system data area, it moves the RAD bootstrap to memory locations 0 through 63 and transfers control to the bootstrap. Then the bootstrap goes through its normal loading procedure (described later in this chapter in "Initial Loading of System Processors").

UPD OPTION (UPDATE)

The UPD option on the SYSLOAD command specifies that a new version of RBM has been made, but that none of the areas on the RAD have changed in size. The option can also be used when changes are made in any of the following input parameters:

- Public Library (PL) FWA
- Resident Foreground FWA
- Nonresident Foreground FWA
- Background FWA

UPD should not be used if any of the RAD areas has changed in size or location. In this case, a complete SYSGEN and SYSLOAD must be performed. Note that a change in the background FWA to increase the total size of background might cause a change in size of the Checkpoint area, which could necessitate a complete new SYSGEN. In this case, a CP AREA TOO SMALL alarm would be output for the user's information.

The System Load Processor reads the bootstrap to determine where the old version of the RBM is located on the RAD and then loads the Monitor Constant Table. The SLP then compares the old load addresses against the new load addresses to determine which programs on the RAD must be reloaded.

The size of the new Master Dictionary must be at least as large as the old Master Dictionary. If it is not, an error message will be output and SLP will continue

If the new version of RBM exceeds the RAD space allocated to the old version, all programs in the System Processor area and all programs that make external references to Monitor service routines (MSR) must be reloaded. (Reloading the System Processor area is necessary because the RBM is the first file in the area.) As the comparison checks are made, a subset of the following messages will be typed on OC:

```
!!RELOAD
  PUB. LIB.
  RES. FGD.
  NONRES. FGD.
  BCKG.
  SP AREA
  MSR/PL USERS AND PL
  NOTHING
```

If any of the following modules are relocated on the RAD, the contents of other affected areas must be reloaded:

Relocated Module	Required Reloading
Public Library requires reloading because its load address has changed.	All programs that reference the Public Library must also be reloaded. None of the system processors use the Public Library, and no system processors would have to be reloaded.
Resident or nonresident foreground was relocated.	The appropriate routines must be reloaded in these areas.
Background was relocated.	All system processor and background user programs must be reloaded. (See "Initial Loading of System Processors" below.)
New RBM version exceeds its allocated RAD file space.	All programs in the system processor area must be reloaded. (See "Initial Loading of System Processors" below.)[†]
TVECT Table load address has changed.[††]	All programs referencing Monitor service routines (MSR) or the Public Library (PL) through the TVECT Table via an external reference must be relocated.

[†]The only areas of the RAD that would never have to be reloaded are the system and user library areas since these areas contain library programs in relocatable binary format.

[††]The TVECT load address will change any time the first word address of the area adjacent to RBM in core has changed.

After these checks are made, The SLP outputs the message

```
!!LOAD RBM PART 2
```

and proceeds to load the overlays as described earlier in the "ALL Option".

After the overlays are loaded, another check is made to see if the overlays did not overflow RBM. If the overlays did overflow into the next area, the following message is output:

```
!!RELOAD
SP AREA
```

After the necessary RELOAD alarms are output for the user's information, the SLP will load the Master Dictionary from the RAD version of RBM and store it into its allocated area in the new version of RBM. The new version of RBM will then be written onto the RBM file, followed by an updated bootstrap in the BOOT file, the starting sector of the system RAD, and the PM device. Finally, the Transfer Vector Table and the RBM Symtol Table will be updated and then rewritten on the RAD.

INITIAL LOADING OF SYSTEM PROCESSORS

For a complete system load, the first processor that is loaded must be the Overlay Loader. The Overlay Loader is coded in a self-relativizing format and is loaded by the RBM Absolute Loader. An entry in the System Processor Dictionary for the Overlay Loader will be made at SYSLOAD time.

The object module of the Overlay Loader will be loaded from the AI device and written into its assigned file. The user must precede the loading of the Overlay Loader with an SY key-in and an !ASSIGN OV=OLOAD,SP control command.

After the Overlay Loader has been loaded onto its permanent file, it is available to load a relocatable binary deck of the RAD Editor onto the RBMOV file of the RAD. The RAD Editor is then executed via an !XEQ command and makes an entry for itself in the System Processor area by means of an !#ADD control command. It then should be copied onto its defined file. At this point, the System Processor area of the RAD contains the Overlay Loader and the RAD Editor, which are the only processors needed to complete the loading of other programs.

PUBLIC LIBRARY CREATION OR UPDATING

The Public Library can be created after the Overlay Loader and RAD Editor have been loaded 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 Library.

RESIDENT FOREGROUND CREATION OR UPDATING

In an initial load the resident foreground files must be defined via the RAD Editor. These files must be in the User Processor area (UP) of the RAD. Also, the parameter 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. On an initial load 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 the resident foreground is loaded on the RAD, 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 OV assigned to its RAD file.

When core is reloaded from the RAD, all newly loaded Public Library and/or resident foreground programs will be loaded and executed, if appropriate (see the description of the RAD bootstrap process at the end of this chapter).

When it is desired to test a new version of a resident foreground program in core before it becomes permanent on the RAD, it can be loaded and executed from the RBMOV file. After the program has been tested, it can be loaded permanently on the RAD using the previously described procedure.

NONRESIDENT FOREGROUND CREATION OR UPDATING

Nonresident foreground programs can be created or updated in their predefined RAD files at any time. The Overlay Loader will read in relocatable binary decks of the nonresident foreground programs, convert their addresses to absolute form, and write them into their defined files. The nonresident foreground files will be located in the user processor area of the RAD, and the definition of the files will be accomplished by the commands to the RAD Editor.

SYSTEM PROCESSOR AND LIBRARY CREATION

The system processors (Extended Symbol, Rad Editor, Basic FORTRAN IV, Concordance, and Utilities) can be loaded or updated by the Overlay Loader from relocatable binary decks. These processors will have their address converted to the absolute locations appropriate for the background area and will be written onto their predefined files in the system processor area. Any processor can then be executed by input of the appropriate !xxx command (where xxx is the file name of the processor).

The System Library will be input in relocatable binary form by the RAD Editor and written in relocatable binary form onto the system library area of the RAD. The construction of several dictionaries in the system library is performed by the RAD Editor.

SYSLOAD ALARMS

In addition to the RELOAD alarms listed previously and those concerned with loading RBM Part 2, the alarms given in Table 30 can be generated and are unique to SYSLOAD.

REBOOTING THE SYSTEM FROM RAD

The system can be rebooted from the RAD by manually causing sector zero of the system RAD to be loaded, or by reading in the RAD bootstrap previously punched on the BO device. The RAD 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 the last cells of the bootstrap and is supplied by SYSLOAD when the bootstrap is written on the RAD or punched out. After the resident portion of RBM is loaded, control is transferred to another bootstrap that loads the remainder of the RAD. This bootstrap functions in the overlay region of the RBM.

Table 30. SYSLOAD Alarms

Message	Meaning	Recovery
CP AREA TOO SMALL	The size of background has changed and/or RAD area allocated for a checkpoint is too small.	To perform checkpoint, SYSLOAD will have to be rerun using the ALL option.
INVALID PARAMETER	An invalid input has been made to the INPUT OPTION request.	Retype either ALL or UPD.
UNPROTECT RAD	One of the write-protect switches has been set on the RAD for an area that SYSLOAD is attempting to modify.	Remove the write protection for the appropriate area.
EOT ON SP AREA	An end-of-tape status has been returned while writing on the SP area. Not enough room has been allocated for the SP area.	A new SYSGEN will have to be run with an increase in the SP area.
EOT ON SD AREA	Same as for SP, except the SD area has overflowed.	Same as for SP.
RDdn FAULT	A nonexistent address has been given for a seek operation.	Check RAD allocation parameters in SYSGEN for allocation of more tracks than exist on this RAD. Repeat SYSGEN and/or SYSLOAD as necessary.
MASTER DICTIONARY OVFLOW	Version of RBM on RAD has a larger Master Dictionary than new version.	Last areas of old dictionary were lost. A new SYSGEN may be necessary.
ALT. TRK. POOL OVFLOW	Too many bad tracks were encountered during the SYSLOAD process.	Some bad tracks will not be in the Alternate Track Pool. A new SYSGEN may be necessary.

The second bootstrap initially inputs the Transfer Vector Table to complete the loading of the resident portion of RBM. Next, 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. The bootstrap then searches the User Processor Dictionary for all files flagged as a resident foreground file. All such files are input, one file at a time, and an inialization 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 the bootstrap. The initialization routine is linked to via the following instruction:

```
RCPYI  P,L
```

It then 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. A provision is made to reboot the system without loading resident foreground. This is accomplished by setting all data keys to -1 just before clearing the "wait" state entered by the initial RAD bootstrap.

The system is then completely rebooted and the bootstrap sets the protection registers, outputs the following messages, and enters a "wait" state.

```
!!AFTER 'WAIT' SET PROTECT 'ON'
!!SET PARITY TO 'INT.'
!!INT. AND 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.

The above messages may be inhibited by setting DATA switch No. 2 prior to execution of the bootstrap. The indicated operations must still be performed, however.

The loading of resident foreground can be inhibited by entering -1 in the data switches before executing the initial bootstrap.

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. In this case Debug must be assigned to an interrupt level higher than any level assigned to the tasks being checked out. During real-time foreground program debugging, no background program may be executed and the background space can be used as an insertion area. The foreground user is able to force an unusual exit from the highest active interrupt level below Debug.

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 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 defines an interrupt level for subsequent Debug use. At this time Debug saves the RBM group code (R:RBMWD) and the register bit (R:RBMB), replaces them with the computed user's group code and register bit, inhibits interrupts, triggers the new Debug level, and exits (resetting inhibit bits) from RBM. The RBM Control Task is now operating at a level where Debug can affect the foreground user's program. After debugging, the foreground user issues the Debug command Q which restores the RBM Control Task 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 B).

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).

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; and if a dump was in progress, the key-in will be honored after the current line is output. 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

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. $NAME, 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
$SQRT
ABC+$SUB1+1492
$SUB1 - $SUB2
```

If the $NAME option is invoked, the user must define an insertion block (see the Debug Define command, below), and the last 180 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 $NAME option is to be used.

The form of the Define command is

```
D [start, end] [,level]
```

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.

level specifies the memory location of the hardware interrupt level if Debug is to be used for foreground. 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

$$\begin{Bmatrix} IB \\ IA \\ IR \end{Bmatrix} \text{ loc, } inst_1, \ldots, inst_n$$

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 Appendix J 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 Appendix J 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 Appendix N.

An example of the insert command is as follows:

```
IB SSUB+1000, 80*SSUB+25, 75A1, 40*SSQRT+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 Appendix L for the calling sequence for a Snapshot command.)

The form of the Insert Snapshot command is

$$\begin{Bmatrix} S \\ SK \\ SS \end{Bmatrix} \text{ loc } [/conditions/] ['message'] [, dump\ requests]$$

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.

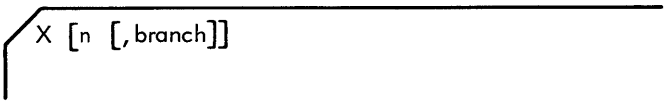
An example of the snapshot command is as follows:

```
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.

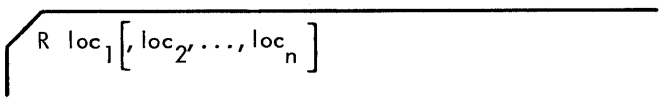
branch is the branch location.

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

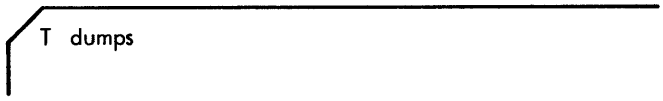


where loc is the memory location.

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	$SUB+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)

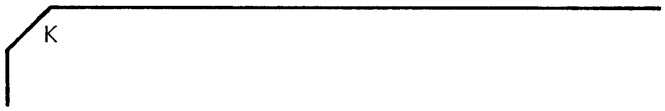
The C command gives control to the Debug input device.

The form of the C command is

K (Keyboard/Printer)

The K command gives control to the keyboard/printer.

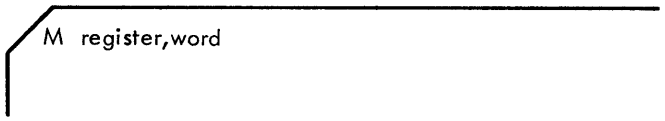
The form of the K command is

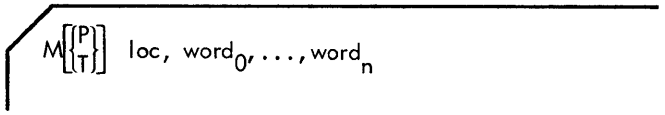


M (Modify Memory)

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. MSSUB+1, 4, 1, SSUB+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, SSUB

   This sets the A register to 0100. Note that an MRP command will change the program address portion of the program status doubleword.

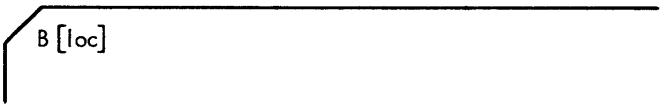
3. MT 149A, RCPYIPA

   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)

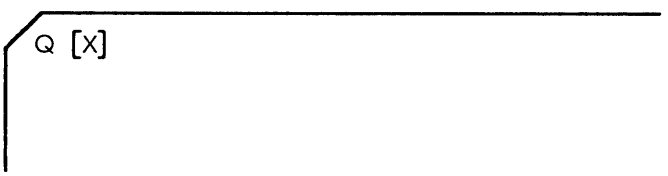
The E command allows the user to force an unusual exit from the highest active interrupt level below Debug. Debug will still have control after this command.

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



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 queuing in Debug (i.e., an unsolicited key-in of Q DEBUG).

APPENDIX A. SIGMA 2/3 STANDARD OBJECT LANGUAGE

INTRODUCTION

The XDS Sigma 2/3 standard object language provides a means of expressing the output of a processor in a standard format. All programs and subprograms in this object format can be loaded by the XDS Sigma 2/3 Overlay Loader. The complete standard object language contains 15 load item types.

An object module consists of the ordered set of binary records generated by an assembly or compilation for later loading. The Overlay Loader has the facility to load and link several object modules together to form an executable program.

The Sigma 2/3 RBM System Absolute Loader can load a single module (absolute subset) to form an executable program. The following load item types from the standard object language comprise the absolute subset:

1. Record Header
2. Record Padding (type 0, subtype 0)
3. Repeat Load (type 0, subtype 1)
4. Unrelocated Load (type 1)
5. Start Module (type 4)
6. End Module (type 5)
7. Load Origin (type 7)

This subset is acceptable input to the resident RBM Absolute Loader and Overlay Loader.

DESCRIPTION OF OBJECT MODULES

GENERAL DESCRIPTION

An object module consists of a set of binary object records, each containing an integral number of load items after a standard three-word record header (see Figure A-1). Each binary record in the module is a 120-byte record.

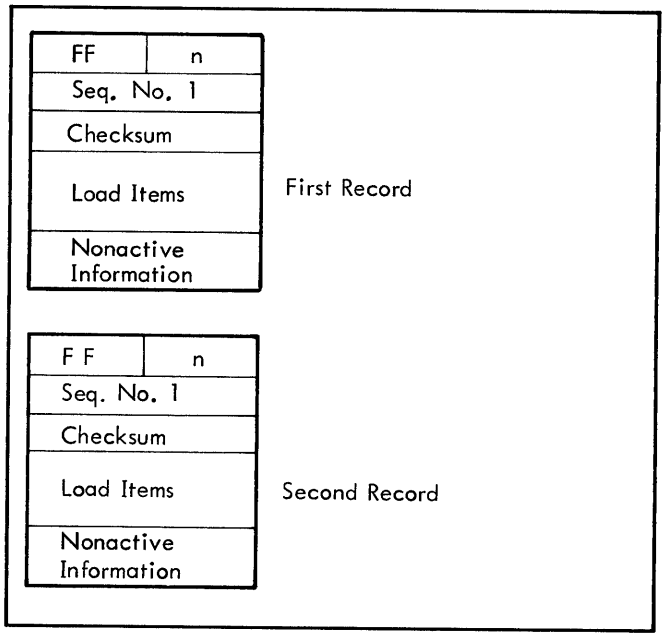


Figure A-1. Typical Object Module of M Records

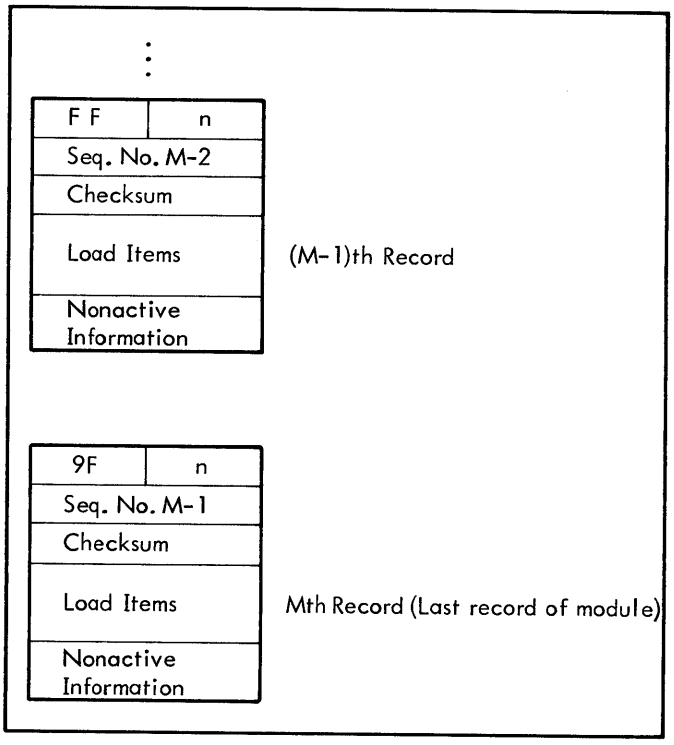
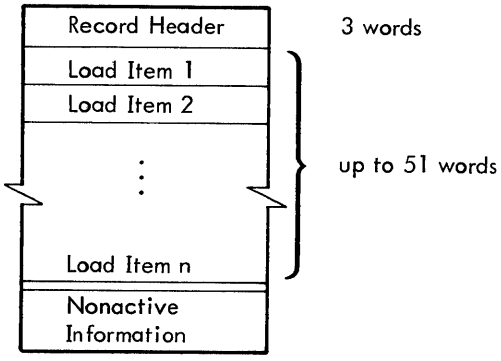


Figure A-1. Typical Object Module of M Records (cont.)

Each load item consists of a header word followed by a variable number of data words. The first load item in an object module is a start-module item and the last item (other than record padding) is an end-module item. There are 15 types of load items, described below.

BINARY OBJECT RECORD FORMAT

Each 120-byte binary record in an object module consists of these parts: Record Header, Load Items, and Nonactive Information in the following arrangement. The Record Header and Load Items are considered the "active" portion of the record.

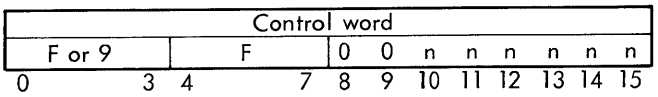


The "active" portion of the record is that information concerning type, sequence number, checksum and binary data usually processed by loaders. The "nonactive" portion may contain sequence or identification information, or it may be empty. It is not processed by the loaders.

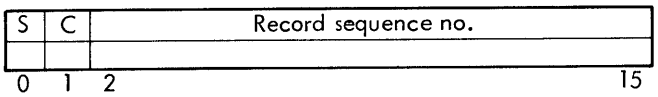
FORMAT OF RECORD HEADER

The first byte of the record header may be either X'F' or X'9'. X'F' denotes that this is a standard record of the object module: X'9' denotes that this is the last record of the object module.

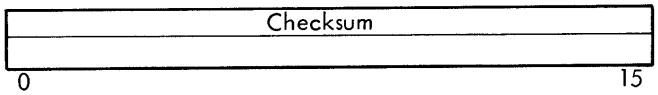
word 0



word 1



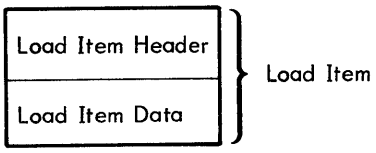
word 2



nnnnnn in the first word is the number of active words in the record, excluding the record header. "Active" denotes data to be processed by a loader. There may be some padding words or sequence information at the end of the record that is not included in the "active" count. The maximum value of n is 51. Note that although the physical record size is fixed at 120 bytes (80 columns of binary data) the number of active words may vary from 3 to 54. This effectively standardizes the reading of binary object records but allows versatility in the generation of active data. The record sequence number starts at 0 and takes on consecutive integer values for all the records in one file. The S bit is a sequence override. If this is a 1, the loader ignores sequence checking for the record. The checksum is an arithmetic sum, with carry, of the n-3 active words after the record header. If the C bit is a 1, the checksum is ignored.

LOAD ITEM FORMAT

Each load item consists of a one-word header and an optional variable-length body of data.



FORMAT OF LOAD ITEM CONTROL (Header) WORD

Every header word has the same general format:

bits 0-3 Type.

bits 4-7 Subtype or control.

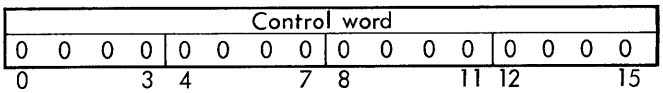
bits 8-15 Number of data words in the load item (excluding item header).

This number plus 1 is equal to the size of the load item. All words of a load item must be contained in the same physical record.

SUMMARY OF LOAD ITEM FORMATS

RECORD PADDING (Type 0, Subtype 0)

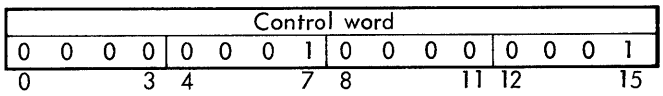
word 0



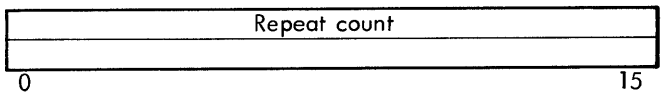
There is no body of data. Padding words are ignored by the loader. The object language allows padding as a convenience for processors.

REPEAT LOAD (Type 0, Subtype 1)

word 0



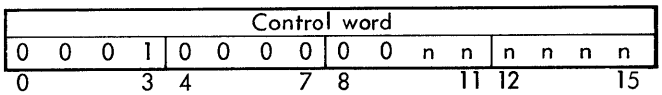
word 1



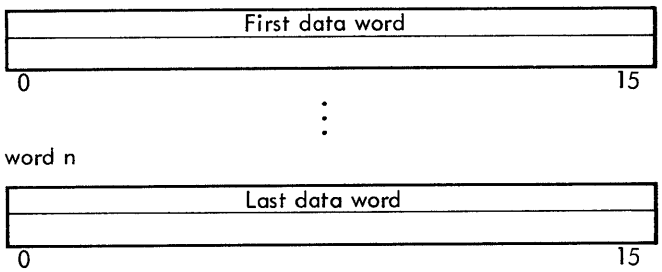
This item repeats the next load item a specified number of times. The load item (type 1, 2, or 3 only) immediately following the repeat load is repeated (i.e., loaded) in its entirety the number of times indicated by the data word.

UNRELOCATED LOAD (Type 1)

word 0



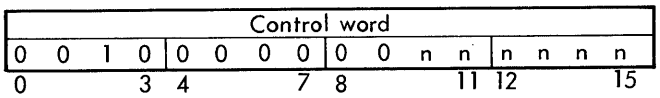
word 1



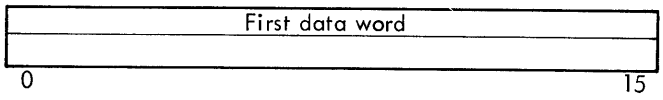
This item loads n words without relocation.

RELOCATED LOAD-MODULE BASE (Type 2)

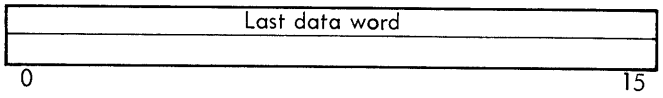
word 0



word 1



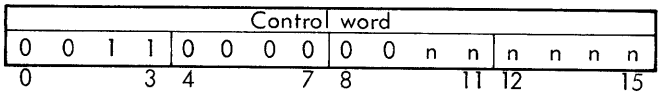
word n



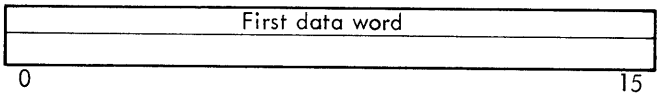
This item loads n words with module relocation. The relocation bias of the current object module is added to each data word in the item.

RELOCATED LOAD-COMMON BASE (Type 3)

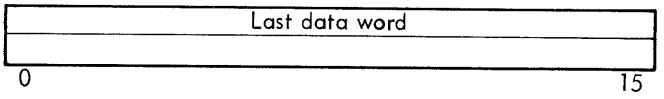
word 0



word 1



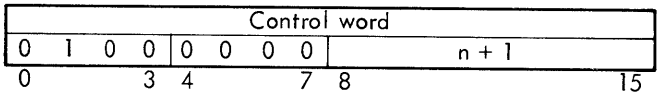
word n



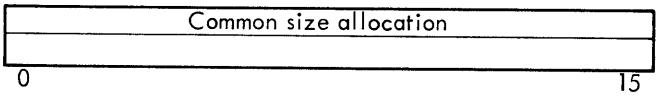
This item loads n words with a common base relocation.

START MODULE (Type 4)

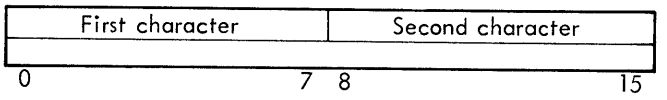
word 0



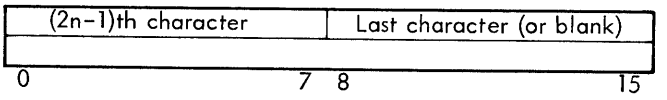
word 1



word 2



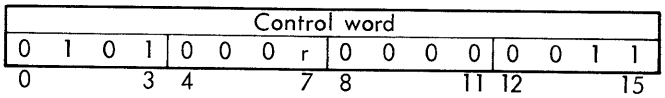
word n + 1



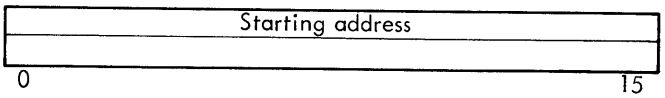
This item identifies the start of the object module. The characters in words 2 through n + 1 are the program name (identification) for the module.

END MODULE (Type 5)

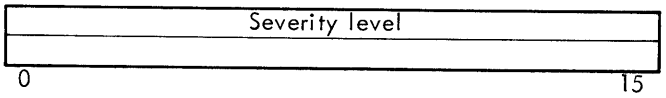
word 0



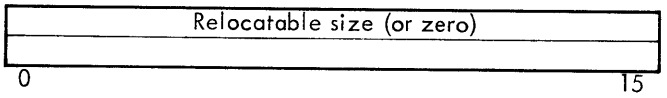
word 1



word 2



word 3



This item identifies the end of the object module. In the control word (word 0), the starting address is defined in bit 7

where

r = 1 indicates absolute starting address.
r = 0 indicates relocatable starting address.

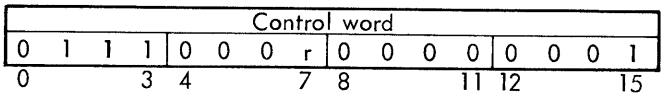
The severity level in word 2 is defined as the highest level reached during processing.

The loader uses the relocatable section size, if present, rather than its own location counter to determine the starting location for the next relocatable section.

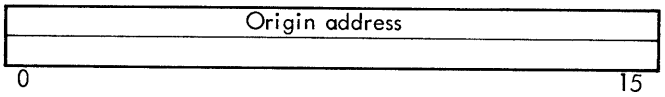
A starting address of absolute 0 indicates there is no starting address for this module.

LOAD ORIGIN (Type 7)

word 0



word 1



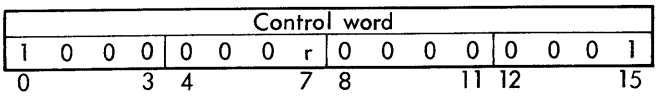
This item sets the origin within the object module. In the control word (word 0), the origin is defined in bit 7

where

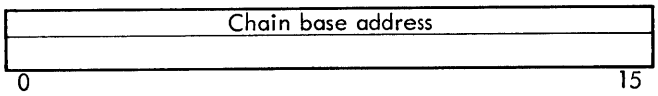
r = 0 indicates relocatable origin.
r = 1 indicates absolute origin.

RELATIVE LOCATION POINTER (Type 8)

word 0



word 1



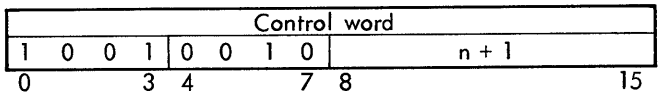
This item establishes the chain base for later chain resolution. In the control word (word 0), the chain base address is defined in bit 7

where

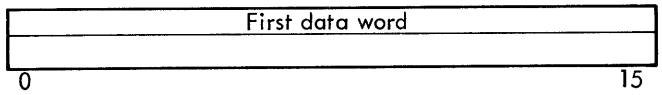
r = 0 indicates a relocatable address.
r = 1 indicates an absolute address.

NAME DEFINITION (Type 9)

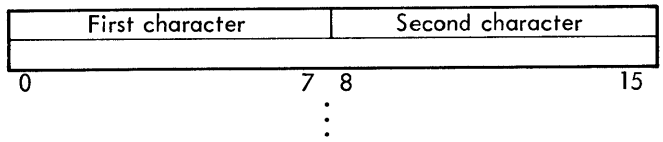
word 0



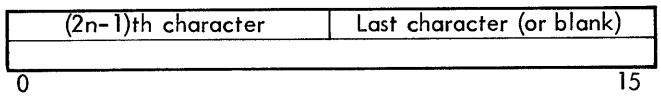
word 1



word 2



word n + 1

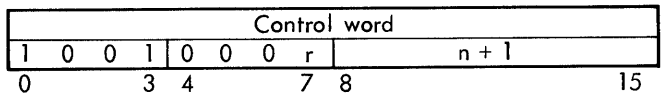


This item identifies a name as a definition within the object module.

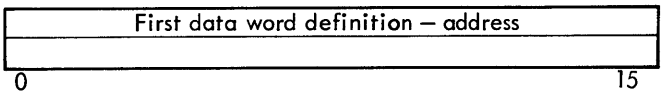
All name definitions immediately follow the start-module item and must precede all other load items. For each name definition, an address definition should appear later in the object module.

ADDRESS DEFINITION (Type 9)

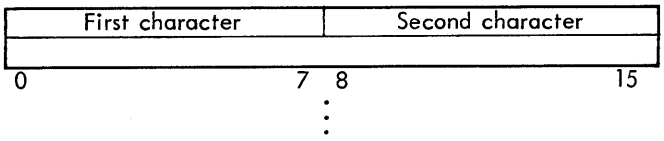
word 0



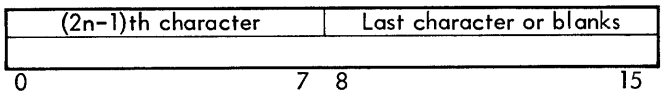
word 1



word 2



word n + 1



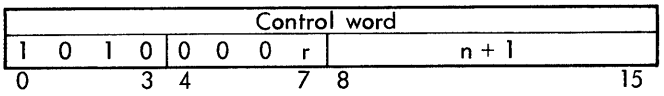
This item associates a location in the module with a definition name (characters in words 2 through n + 1) for other modules to reference. In the control word (word 0), the definition address is defined in bit 7

where

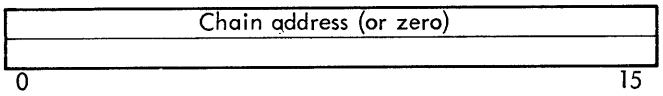
r = 0 indicates relocatable definition address.
r = 1 indicates absolute definition address.

EXTERNAL REFERENCE (Type A)

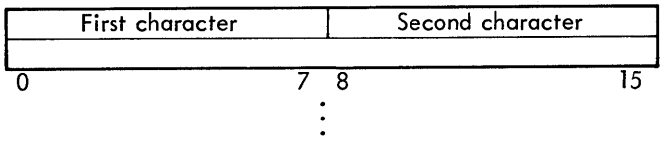
word 0



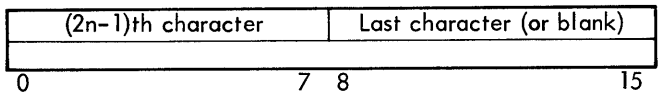
word 1



word 2



word n + 1



This item states a name (characters in words 2 through n + 1), defined in another module, whose definition address must be inserted in a chain of locations within the module. In the control word (word 0), the chain address is defined in bit 7

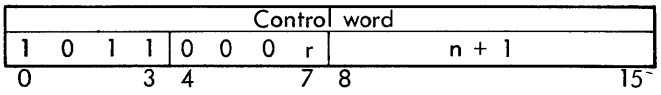
where

r = 0 indicates a relocatable chain address.
r = 1 indicates an absolute chain address.

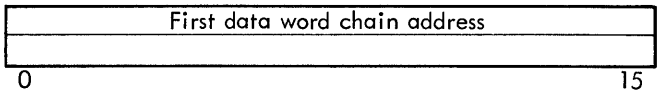
Note: If there is no chain address, the reference address is zero and is used for library searching purposes only.

SECONDARY REFERENCE (Type B)

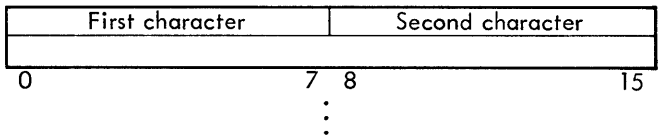
word 0



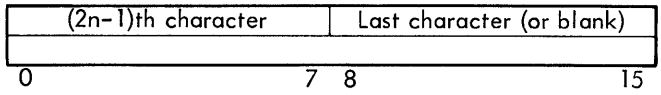
word 1



word 2



word n + 1



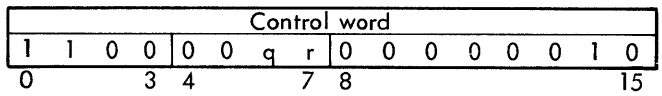
This item states a name (characters in words 2 through n + 1), defined in another module, whose address may be inserted in a chain of locations within the module. This item is identical to type A, above, except that it does not force loading of the routine from the library. In the control word, the chain address is defined in bit 7

where

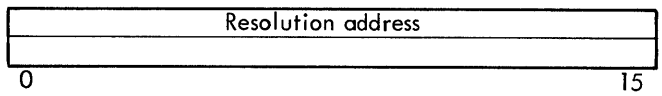
r = 0 indicates a relocatable chain address.
r = 1 indicates an absolute chain address.

ADDRESS LITERAL CHAIN RESOLUTION (Type C, subtypes 0, 1, 2, and 3)

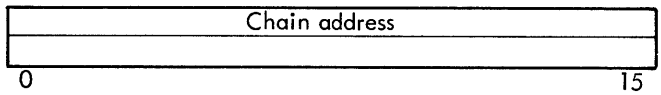
word 0



word 1



word 2



This item defines a location within the module (called the resolution address) whose address must be inserted in a chain of displacement fields within the module. In the control word, the chain address is defined in bit 6

where

q = 0 indicates a relocatable chain address.
q = 1 indicates an absolute chain address.

The resolution address is defined in bit 7

where

r = 0 indicates a relocatable resolution address.
r = 1 indicates an absolute resolution address.

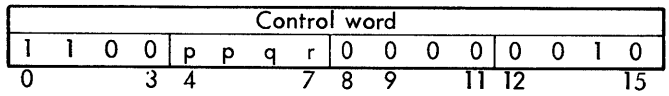
An address literal chain is a threaded list of forward references to a single location in a program. The definition value (called the resolution address) can be output as an address literal chain resolution (Type C, subtypes 0, 1, 2, and 3). The chain address points to the beginning of the threaded list which is terminated by an absolute zero value. The resolution address and the chain address may be absolute or relocatable.

Note: Because the terminator of the chain is zero, no program may have an address literal chain whose last link is at absolute zero (i.e., the item would reference zero and would thus appear to terminate the chain).

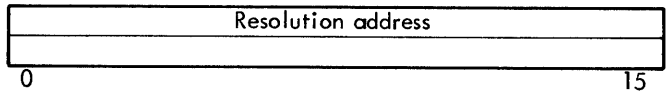
Note that external reference (REF) (type A) and secondary reference (SREF) (type B) chains are structured in the same manner, but resolved by the loader using an external definition value (type 9).

DISPLACEMENT CHAIN RESOLUTION (Type C, subtypes 6, 7, A, and B)

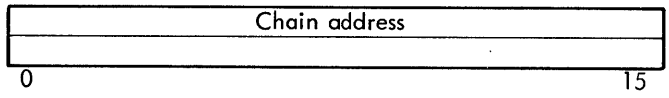
word 0



word 1



word 2



This item defines a location (called the resolution address) within the module whose relative displacement must be inserted in a chain of displacement fields within the module. In the control word, the displacement chain is defined in bits 4-5

where

pp = 01 indicates that an indirect bit is not set in each instruction in the displacement chain.

pp = 10 indicates that an indirect bit is set in each instruction in the displacement chain.

q = 1 always indicates absolute displacement of the last item in the chain (relative to the chain base declared in item type 8).

The resolution address is defined in bit 7

where

r = 0 indicates a relocatable resolution address.
r = 1 indicates an absolute resolution address.

When forward references occur during one-pass processing, and the possibility of resolving the reference by a definition or literal may occur within 255 locations, the 8-bit displacement field of the instruction may be used to form a displacement chain. The item types 8 (relative location pointer – establish chain-base) and C (displacement-chain resolution) must be used together to resolve the chain by substituting actual displacements determined at load time.

In the creation of a displacement chain, the pointer in the type 8 item defines the relative location in the program to be established as the chain base. Each new type 8 item can define a new chain base. The values in the displacement field of the instructions included in any given displacement chain refer to the absolute displacement of that instruction relative to the currently established chain base; e.g., if the chain base is established to be X'100' and an instruction is located at X'125', the displacement of that instruction for purposes of the displacement chain is X'125'-X'100' or X'25'. This point is emphasized since the loader will use this displacement only to determine the final displacement of the instruction relative to the location of literal or target locations.

When the displacement chain connects instructions that reference a literal or a specific target location within range of the chain base (e.g., LDA=3 LDA=LAB, B XR), no indirect bit is set in each instruction (pp = 01 in Header – Type C).

When the chain connects references to an external symbol or forward reference whose value will be given in some literal within range of the chain base, pp is set to 2 in the type C header, to set the indirect bit in each instruction in the chain (e.g., LDA X, which will be resolved as LDA *$+n, where n is the displacement of ADRL X relative to the instruction).

The chain base address (in the type 8 item) may be declared as an absolute or relocatable value. The resolution address (first data-word of a Type C item) is the address of the target location or literal expressed as a location, and not as a displacement on the chain base. Note that although the resolution address is defined at this point, the value of the literal at that resolution may not be defined until later. In fact, it may be an element of an address-literal chain (type C) or external reference chain (type A). The address-literal or external chain resolution is independent of the displacement chain resolution.

The chain address given in the second data word is the absolute displacement of the last item in the chain, relative to the chain base declared in type 8 (e.g., if the effective chain base were X'1000' and the value of the chain address were X'20', the last item of the displacement chain would be located at X'1020').

A separate displacement chain will be created for each unique variable in a given displacement region. Thus, many displacement chains may be built using the same chain base. As a matter of fact, the chain base may not be changed until a displacement chain resolution item has been output for each displacement chain. An unresolved displacement chain is a serious error condition in the output, and is unacceptable for execution.

The format of the displacement chain is described in the example in Figure A-2.

Example: Let a chain base be declared at 109(R). (Numbers given are decimal.) It is assumed that the ADRL for XLB will be ultimately loaded at 140(R). Note that the displacement field of each instruction before resolution is a pointer to the location of the next item in the threaded list relative to the chain base.

Relative Location Counter	Symbolic	Displacement From Chain Base	Displacement Field of Instruction Before Loading	Displacement Field of Instruction After Resolution
110	LDA XLB	1	00 (end of chain)	30 (140–110)
125	STA XLB	16	01	15 (140–125)
134	CP XLB	25	16	06 (140–134)
136	STA XLB	27	25	04 (140–136)
140				
	Item Type C, Displacement Chain Resolution			
	Resolution Address 140(R)			
	Chain Address 27(A)			

Figure A-2. Displacement Chain Format

APPENDIX B. SYSTEM ZERO TABLE AND CONSTANTS

Table B-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	R:IOP	Pointer to 8-word IOP Table.
9 : 63	9 : 3F		Standard Constants for Foreground, Monitor, and Background Use (see Table B-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 : 194	A8 : C2		Real-Time Foreground User Storage (reserved for foreground communication between foreground and background or for address literals or constants).

Table B-1. Monitor Zero Table (cont.)

Address		Name	Purpose and Assignment
Dec.	Hex.		
195	C3		Power OFF.
196	C4		Power ON.
197	C5		Integral IOP timeout.
198	C6		Watchdog Timer timed out.
199 : 231	C7 : E7		Monitor Service Routines Transfer Vectors (see Table 7 for list).
232 : 251	E8 : FB		Monitor Constants (see Table B-3).
252 : 255	FC : FF		Counter Interrupt Locations (optional).

Table B-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 B-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 B-3. Monitor Constants

Address			
Dec.	Hex.	Name	Purpose
226	E2	K:IOCS	Pointer to IOCS Tables.
227	E3		Reserved for Monitor use.
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	K:VRSION	RBM Version.

Table B-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	K:MTMP	Size of Nondynamic Storage, in Words (6).
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.

[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:IOCS EQU X'E2').

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 C. RBM SYSTEM ABORT CODES

The abort codes given in Table C-1 are the standard abort codes output by the Monitor, Basic FORTRAN IV Compiler, Extended Symbol assembler, Utility Subsystem, and RAD Editor (see also supplementary control command diagnostics in Appendix D).

OVERLAY LOADER ABORT CODES

The abort codes given in Table C-2 will be output by Overlay Loader which will then exit via a call to the RBM routine M:ABORT.

LOADER I/O ABORT MESSAGE

The I/O abort message has the following format, followed by the message "ABORT IO location":

oplb device type and number diagnostic

where

oplb is the operational label of the device or file on which the error occurred.

device type and number pertain to the operational label.

diagnostic is an error diagnostic corresponding to an I/O completion code.[t]

The following diagnostics may be used:

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
ABORT  IO       3F4C
```

where

BI is the oplb.

MTD0 is the device type and number.

END OF TAPE is the diagnostic.

3F4C is the ABORT IO location.

[t] See Table 10, "I/O Completion Codes", in Chapter 4.

Table C-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.

Table C-1. RBM Abort Codes (cont.)

Code	Meaning
DE	Debug not resident when requested.
ER	Operator-recognized error condition.
ES	FORTRAN library abort†.
FC	Illegal FORTRAN control card.
FS	FORTRAN abort†.
GO	Irrecoverable error on output to the GO file when using a !REL command.
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.
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).
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.

Notes: 1. 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 in with an indication (">" will precede the command) that they have been ignored.

2. If integral IOP timeout occurs, RBM checks foreground mailbox X'C5' for a watchdog receiver. If a receiver is specified, RBM branches to it; otherwise, RBM halts with the address of the interrupt in the accumulator. An integral IOP timeout indicates hardware difficulties.

Table C-2. 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.
A9	Error in closing the RBMID file.
BB	Cannot use RS' op label because it is already used by Overlay Loader.
CM	A COMMON displacement or size larger than that stipulated on the !OLOAD command or in a start item was detected. (Background abort only.)
CR	A non-COMMON item was relocated into COMMON. This condition only occurs when an actual data item is to be stored into COMMON.
DS	The same identifier was used to name two different segments.
EF	An illegal end-of-file was detected.
IT	An illegal item type was detected.
LI	The library files cannot be loaded because of incorrect construction of the library.
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	Overlay Loader was unable to write the Public Library, the LIBSYM, or the TVECT files onto the RAD.
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.
SD	Next segment of the Overlay Loader cannot be loaded.
SE	Input ROM had an error severity level greater than zero.
SG	Format or parameter error was detected on a !$SEG command.
SL	The length of a segment was excessive, (see !$ROOT and !$SEG commands for maximum segment size).
TO	There was a table overflow. Decrease the size of the program or reduce the number of external symbols.
UN	The number (on the !$SEG card) of the segment to which this one is attached has not been defined.

APPENDIX D. CONTROL COMMAND DIAGNOSTICS

The following error messages may appear on the background DO device as a result of an error condition detected by the JCP. These diagnostics supplement the abort or attend error codes printed by the 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	WEOR, REWIND, UNLOAD, FBACK, FSKIP, RBACK, RSKIP
.RAD TEMP OVERFLOW	DEFINE, default assignments for system processors

The following table should be used to determine the standard assignments for an installation's RBM operational labels and to determine which operational labels, if any, should be suppressed by being assigned to file 0. The RBM operational labels are defined under the !ASSIGN command in Chapter 2.

RBM and Processors \ RBM Operational Labels	Device Number 1	CC	SI	UI	AI	BI	BO	UO	LL	DO
RBM	Read/Write unsolicited key-in	Read Control Commands			Read Absolute Binary	Read Object modules with !REL command			Write Control Command Images	
XSYMBOL		[Read Control commands]	Read Source Statements				Write Reloc. Binary		Used for CC Diagnostics	Write XSYMBOL Error Messages[tt]
Concordance			Read Source Statements							Write Concordance Error Messages[tt]
Basic FORTRAN IV			Read Source Statements				Write Reloc. Binary			
Math Library										Write Library Error Messages
Overlay Loader		Read Control Commands								Write Map, Loader Error Messages and Control Command Images[tt]
RAD Editor		Read Control Commands				Object Module Input to System and User Libraries	Output Copies of Object Modules from System and User Libraries			Write Error Messages, Control Commands and operator key-ins
Utility Executive		Read	Read Control Commands							Write Utility Error Message and Control Command Images[tt]
Utility Copy[t]			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		

[t] May use any op label for output.

[tt] Suppressed if assigned to same device as LO.

RBM and Processors \ RBM Operational Labels	LO	LI	PM	OC	X1	PI	OV	X2	X3	S2	GO	X4	X5
RBM			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	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		Read Reloc. Binary Library File		Operator Communications	Contains Symbol Table for each segment	Read OLOAD Overlays	Write Core Images				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 F. CHARACTER-ORIENTED COMMUNICATIONS (COC) EQUIPMENT HANDLER

This appendix describes the interface of RBM with the Xerox character-oriented communications (COC) equipment.† The COC equipment provides communication between Sigma 2/3 real-time programs and various terminal devices. The COC consists of a controller and from one to eight attached line interface units, with each unit containing from one to eight send-and-receive modules. The Sigma 2/3 RBM can accommodate one COC, which gives the user up to 64 lines each with send-and-receive equipment. The terminal devices supported (one per line) can be Teletype Models 33, 35, or 37. Other terminals can be connected but they must use ANSCII control codes, and all editing must be done by the user program.

The computer requirements for use with the COC equipment are as follows:

1. RBM with at least 16K of core memory.
2. One buffered input/output channel dedicated to the COC controller.
3. Two external interrupts dedicated to the COC controller.
4. External interface feature.

DESCRIPTION OF COC PACKAGE

The COC software package allows messages to be communicated via the character-oriented equipment, and consists of two sections – M:COC and RCOC.

M:COC M:COC is a Monitor service routine that initiates all read, write, and control operations. It is part of the RBM overlays and requires no modification by the user before use. (M:COC is described in detail in Chapter 4.)

RCOC RCOC consists of the following tasks and tables that make up a resident foreground program:

1. An initialization routine.
2. A real-time task connected to the input interrupt of the communications controller, which edits and translates input characters, echoes the characters if required and forms input messages.
3. A real-time task connected to the output interrupt of the communications controller, which transmits output messages and editing characters at end-of-message (EOM).
4. Conversion tables (ANSCII to EBCDIC, and vice versa).
5. An input buffer (overlays the initialization routine).

RCOC must be assembled separately for each installation unless the default cases for the installation specific assembly parameters agree with the parameters desired. The assembly parameters are as follows:

1. The device number of the COC (buffered input/output) (default = 7).
2. The COC number (direct input/output) (default = 0).
3. The input interrupt level (even number of the even-odd pair) (default = 110). The output interrupt level is assumed to be the odd number.
4. Number of lines used (n), where all line numbers 0 to n-1 are assumed to be used (default = 1).

COC OPERATION

RCOC is a resident foreground program and must reside on either the SP or UP area of the RAD. It is read into core memory and operated whenever RBM is rebooted. The RCOC initialization routine turns on all transmitters and receivers, arms and enables the input and output interrupts, initiates input from the COC controller into a wraparound buffer, and exits. At this point, all lines are set to the "disconnected" status, ready to be connected and used by the real-time programs. Input is initiated and an input interrupt is generated for each character input, but the data are ignored until the line is connected and a read request is given.

All line-control and read-or-write operations are initiated by calls to M:COC. A request to read merely causes the line status to be set to "read", which in turn causes the input interrupt routine to accept input from that line and build the input message in the user's buffer. A request to write causes M:COC to turn on the transmitter and transmit the first character in the user's buffer. Thereafter, an output interrupt is generated once each "output word time" (i.e., once each time the transmitter can transmit). The output interrupt routine transmits characters from the user's buffer until the entire message is sent and then turns off the transmitter.

As each input or output message is completed, the status of the line is set to "message complete" and an EOM Receiver (if present) is operated at the input or output interrupt level.

† See Xerox Sigma Character-Oriented Communications Equipment Reference Manual, Publication 90 09 81, for a description of the equipment involved, the possible configurations, and the various uses for the equipment.

The receiver should trigger the requesting task and return to the location contained in the L register.

AUTOMATIC DIALING

If the Automatic Dialing Equipment (ADE) is included, real-time tasks can dial a terminal and connect it to a predetermined COC line. The ADE is a multiunit controller that controls up to 16 dial positions. It requires a dedicated buffered IOP channel.

The dialing operation can be accomplished via M:IOEX. A TDV should first be performed to ensure that the dial position is available. Then an SIO 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 bytes per word; each digit occupies 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.

RESTRICTIONS

The priority of the input/output interrupt pair must be higher than any program using its services via M:COC and should also be higher than other real-time programs with long execution times. If a program with a higher interrupt priority runs for a long period of time, the input buffer may become filled and data may become lost. The output data would be delayed but no data would be lost.

All COC lines (i.e., assembly parameters) are assumed to be operational. The RCOC initialization routine will loop, attempting to turn the receiver on for a nonexistent COC line number.

If automatic dialing is included, the user must include M:IOEX during SYSGEN and must input the dialing positions as XX type devices.

APPENDIX G. SYSGEN AND ASSEMBLY TIME OPTIONS

The optional RBM capabilities below are obtainable as a package in response to the SYSGEN query INC. MISC. At least 100 (decimal) additional resident core memory locations are required.

HEXADECIMAL CORRECTOR CARDS

Patches may be loaded at execution time for either the Monitor itself or any user program. All corrector 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 are loaded from CC in one of two ways:

1. Following a HEX control command.

2. Following an unconditional H key-in.

All corrector decks are terminated by an EOD control command. To patch relocatable programs, a bias card may be used. Its form is

+bbbb

where bbbb is the bias and the following correctors are loaded relative to that location. Any value (on a corrector card following the bias card) preceded by a plus (i.e., $+cccc_i$) will have the bias added to it.

To patch program segments,[t] Data Switch 0 must be placed in the "1" state. This causes the RBM to type "BEGIN SEG xx" (where xx is the segment number; XX = 0 for the root) and go into an idle state after each segment is loaded. Correctors can then be loaded to the segment following an H key-in. An S key-in will cause RBM to resume operation. The ability to type the message "BEGIN SEG xx" is determined when RBM is assembled and is not related to the inclusion of the"MISC." routines.

THREE-CHARACTER PROCESSOR SEARCH

An assembly time option exists for the Job Control Processor (which does not increase resident RBM) to identify a processor from the first three characters input.

When the Control Command Interpreter encounters a processor request such as !XSYMBOL, a search is first made of the system, then the user processor area, to locate the file whose name matches the requested processor exactly. Normally, if this search fails, the Monitor aborts the job. However, if this assembler option has been selected, the request is then truncated to three characters (i.e., !XSY) and the search of the system and user processor areas is repeated. Thus, if Extended Symbol has been defined on the system processor area of RAD as the three-character name !XSY, either a request of !XSYMBOL or !SXY will locate the system processor.

---

[t] An optional assembly parameter in the RBM subtask S:LOAD. This parameter does not increase RBM.

APPENDIX H. MEMORY REQUIREMENTS

CORE SPACE REQUIREMENTS FOR RBM

The minimum RBM system (which would consist of keyboard/printer, paper tape, and RAD I/O routines, and a minimum number of RAD device-files and operational labels) requires about 4300_{10} cells for the Real-Time Batch Monitor and all its tables. This minimum core space requirement will increase as handlers are added for additional peripherals, as additional optional software routines are chosen (see Table H-1) during SYSGEN, and as additional device-files, operational labels, or Public Library DEFs are allocated during SYSGEN. The following table indicates the approximate core space requirements for the additional routines. Unless otherwise indicated, these number are only approximate and have been rounded to the next higher multiple of 25.

Table H-1. Core Requirements for Additional Software

Handler or routine	Approximate size (decimal)
Multiply/Divide Simulation Software	175
Power Off/On	196
M:IOEX	188
Job Accounting	216
Line Printer Handler	79
Card Reader Handler	2 (exact size)
BCD Option for Card Reader	2 (exact size)
Magnetic Tape Handler	208
Card Punch Handler	2 (exact size)
BCD Option for Card Punch	2 (exact size)
Each additional RAD Device File	15 (exact size)
Each additional Operational Label	2 (exact size)
Each Public Library DEF	2 (exact size)

Hence, the resident core space requirements for RBM vary from 4300 to 6200 cells, depending upon the user's configuration. If background processing is desired, the user must allocate at least 3800 cells for background to accommodate the RBM Job Control Processor which executes in the background space.

CORE SPACE REQUIREMENTS FOR THE RBM PROCESSORS

The minimum background space necessary to individually load with the Overlay Loader program and to execute all the RBM Processors is 7K cells ($1K = 1024_{10}$). The largest processor here is Basic FORTRAN IV, which requires 7K cells when it is loaded by the Overlay Loader. FORTRAN programs of reasonable size can be compiled in 7K of background. Extended Symbol can be loaded in a minimum of 6.25K of background, and a program of approximately 1200 to 1800 instructions could be assembled in this minimum space. The other RBM processors can all be loaded and executed in less than 6K of background.

RAD SPACE REQUIREMENTS

Table H-2 gives the allocations for the system areas of the RAD, if a user chooses not to override the default case. The following discussion assumes a 360-byte-per-sector RAD.

Table H-2. RAD System Area Requirements

Area	Size	Comments
Checkpoint	n sectors	n=size of background (in sectors).
System Processor	30 tracks	Sufficient to contain all RBM processors plus RBM.
System Library	9 tracks	Sufficient to contain two versions (extended and basic) of Math/Run-Time Library.
System Data	14 tracks	RBM files.

Note that this leaves approximately one spare track in the system data area. However, if a Public Library is included, the file LIBSYM must be added to the system data area. Hence, the system areas and the checkpoint area will normally consume about 45 tracks of the RAD. (The smallest Xerox RAD, .75 megabyte, has 128 tracks.) The only other area used by the system is the background temp area. The processor that normally requires the largest background temp area is Extended Symbol. Extended Symbol normally requires the background temp area to be split into three

scratch files, called X1, X2, and X3.[t] File X1 is a compressed file and contains the user's source deck (about 12 source cards can be compressed into one RAD sector). File X2 contains the user's source deck in an encoded form (normally about 36 source cards can be stored in one RAD sector on X2). File X3 is only used if the program being assembled contains local symbols. Normally, the RAD space required for X3 is insignificant compared with X1 and X2. Hence, to assemble a 5000-card source program, approximately 35 tracks of background temp area would be required. Thus, if a user wants to have all the system processors and a complete system library stored on the RAD, and wants to allocate enough background temp area to assemble about a 5000-line source program, approximately 80 tracks of the RAD would be used.

---

[t]The Job Control Processor will automatically divide the total background temp area into three scratch files upon encountering an !XSYMBOL command. The total area is divided amoung the X1, X2, and X3 files according to the following ratios:

X1:X2:X3 = 90: 30: 3

The user can override these default allocations by inputting a !DEFINE command prior to the !XSYMBOL command.

APPENDIX I. CALCULATING THE RBM SIZE

To calculate the size of RBM (RBM LWA) before a SYSGEN, add the base value of F86 or 3980:

8 x number of I/O channels

2 x number of definitions in the Public Library

4 x number of entries in nonresident foreground queue

4 x number of Master Dictionary entries

1 x number of entries in Alternate Track Pool

10 x number of RAD/disk pack devices

To this figure add the following:

$C4_{(16)}$ or $196_{(10)}$ cells if a Y response to INC. POWER ON/OFF

$AD_{(16)}$ or $173_{(10)}$ cells if a Y response to INC. MUL/DIV SIM.

$BB_{(16)}$ or $188_{(10)}$ cells if a Y response to INC. M:IOEX

$D0_{(16)}$ or $216_{(10)}$ cells if a Y response to INC. CLOCK ONE

$10_{(16)}$ or $16_{(10)}$ cells if a Y response to INC. DEBUG

$56_{(16)}$ or $85_{(10)}$ cells if a Y response to INC. MISC.

2 or $2_{(10)}$ cells if a Y response to INC. C.O.C.

Add to this amount the number given below (see Table I-1) if the corresponding device type is included in the SYSGEN parameter DEVICE FILE INFO:

To this sum, add two cells for each background or foreground operational label.

Since SYSGEN attempts to store whatever optional routine of tables it can into the unused interrupt locations, the size of the unused interrupt region can generally be subtracted from this accumulated sum. The size of this area can be determined by subtracting the value input for the SYSGEN parameter MAX. INT. LOC from $18F_{(16)}$ ($399_{(10)}$). However, this figure will be less than the true size of RBM since not all of these unused interrupt locations can be used

Table I-1. Device Type Table Allocations

Device	Size			
	First Input		Additional Inputs	
	Hex.	Dec.	Hex.	Dec.
KP	1F	31 (required)	F	15
LP2	1E	30	E	14
LP8	4F	79	B	11
CR4	1B	27	B	11
CP1	1F	31	F	15
CP3	96	150	86	134
Any magnetic tape[t]	D0	208	B	11
PT	1F	31	F	15
PL	19	25	B	11
RD[tt]	-	-	14	20
XX	6	6	6	6

[t] Add two cells to the first input if magnetic tape is BCD.

[tt] The default case for background is nine RD files.

APPENDIX J. 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
```

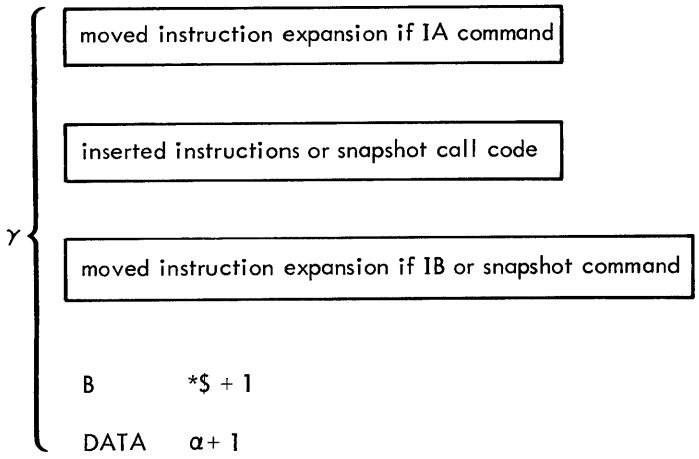
In the above expansions, α is the location (point) of the insertion and op has the appropriate settings for the incrementation and inversion bits.

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.

APPENDIX K. DEBUG INSERTION STRUCTURE

An insertion at location α will result in the following:

```
α    B       *β
β    DATA    γ
```



where β is one of the Debug cells in the zero table and γ is an area in the insertion block.

APPENDIX L. 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):

```
a1      DATA            D:SNAP
a2      DATA            block
        instruction that was at location α
entry   WD              X'FC' (foreground only)
        STA             *a2
        RCPYI           P, A
        B               *a1
        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
```

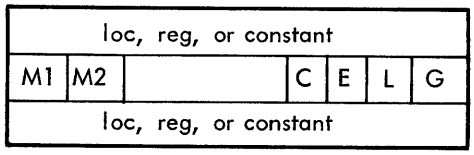
where

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 12) designates equal comparison.

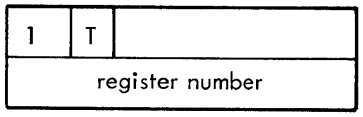
L (bit 14) designates less than comparison.

G (bit 15) designates greater than comparison.

A logical operator occupies one word:

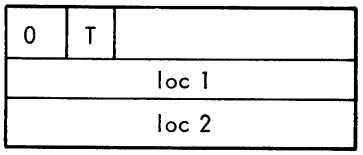
- 0 logical or
- 1 logical and

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.

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.

A

B

C

D

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.

M

N

O

P

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.

Q

R

S

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numberical sequence.

T

U

V

W

X

Z