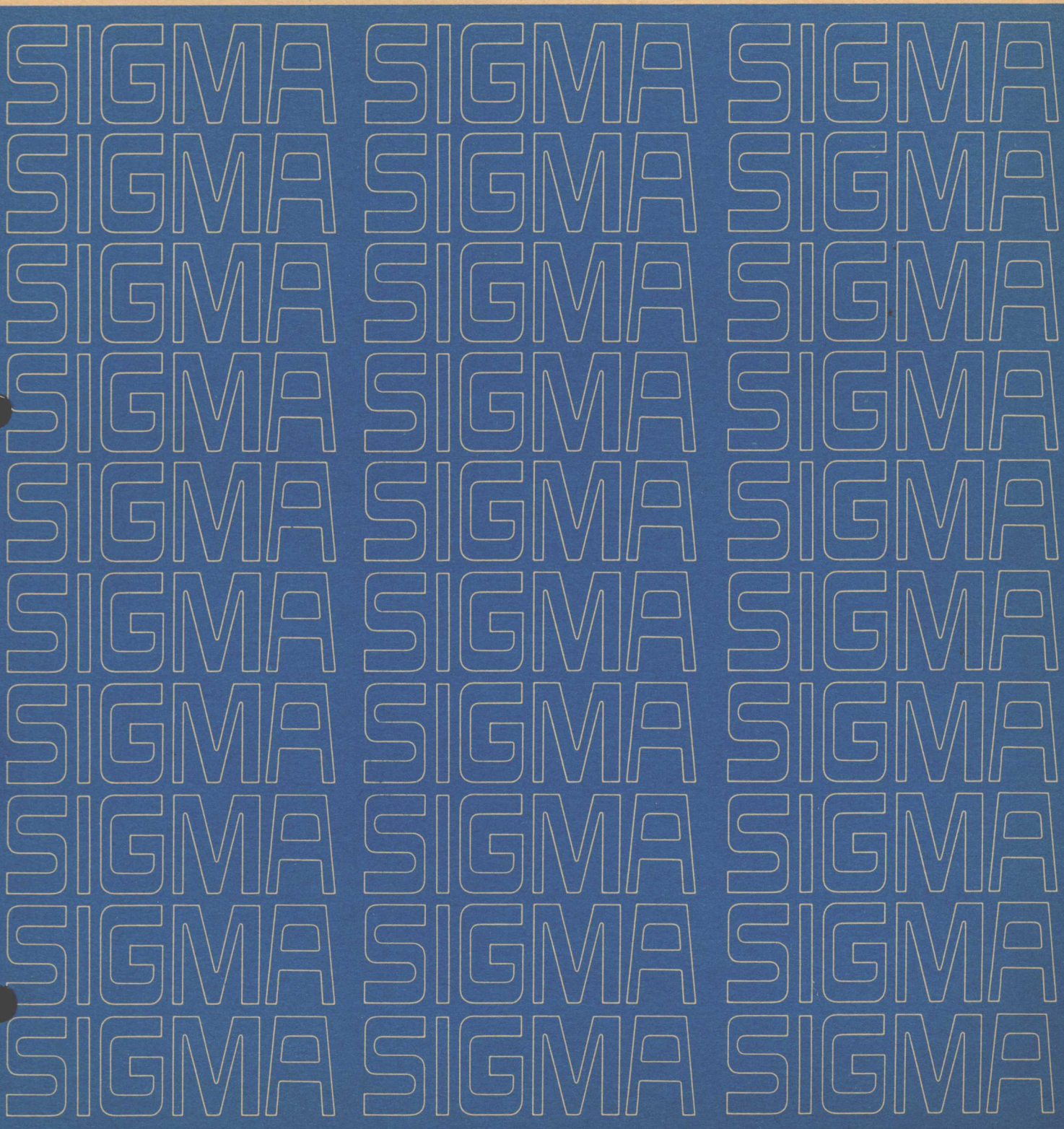
SDS

Scientific Data Systems
A XEROX COMPANY

SDS SIGMA 2/3 BASIC CONTROL MONITOR

Reference Manual



Price: $3.00

BASIC CONTROL MONITOR REFERENCE MANUAL

for

SDS SIGMA 2/3 COMPUTERS

90 10 64C

August 1969

SCIENTIFIC DATA SYSTEMS A XEROX COMPANY/701 South Aviation Boulevard/El Segundo, California 90245

Printed in U.S.A.

REVISION

This publication, SDS 90 10 64C, is a revision of the SDS Sigma 2 Basic Control Monitor Reference Manual, SDS 90 10 64B (dated March 1969). A change in text from that of the previous manual is indicated by a vertical line in the margin of the page. Most of the revisions consist of updating Sigma 2 documentation and software to apply to Sigma 2/3 documentation and software. The symbol ③ is used in the margin throughout this manual to indicate capabilities limited to users of SDS Sigma 3 hardware.

RELATED PUBLICATIONS

Title	Publication No.
SDS Sigma 2 Computer Reference Manual	90 09 64
SDS Sigma 3 Computer Reference Manual	90 15 92
SDS Sigma 2/3 Basic Control Monitor Operations Manual	90 15 06
SDS Sigma 2/3 Symbol Reference Manual	90 10 51
SDS Sigma 2/3 Basic FORTRAN/Basic FORTRAN IV Reference Manual	90 09 67
SDS Sigma 2/3 Basic FORTRAN Operations Manual	90 10 61
SDS Sigma 2/3 Basic FORTRAN/Basic FORTRAN IV Library/Run-Time Technical Manual	90 10 36

CONTENTS

APPENDIXES

ILLUSTRATIONS

TABLES

1. INTRODUCTION

SYSTEM FEATURES

The SDS Sigma 2/3 Basic Control Monitor is a standard software system available for use with Sigma 2/3 computers and has full real-time capability with some provision for batch processing in the background. The Basic Control Monitor[†] can partition memory for simultaneous residency of real-time foreground tasks, Monitor resident space, and batch (background) operations.

The real-time requirements of the Monitor are satisfied with the following features:

- All Monitor service routines are reentrant.
- All background requests are serviced below the priority level of the real-time foreground.
- Very low Monitor overhead is required in the processing of input/output requests.
- Interrupts are not inhibited for more than 100 CPU microseconds.[††]

Regardless of whether the system is used for foreground/background multiprogramming, the BCM is capable of handling, independently and concurrently, up to 132 real-time foreground tasks, each with a unique priority level.

When multiprogramming with foreground/background, the foreground has access to all privileged instructions in Sigma 2/3 computers. The background is checked by both hardware and software to provide complete protection of the foreground programs for both core memory and peripheral operation.

The BCM allows the user to assemble, compile, or perform data processing in the background (concurrent with foreground operations) to absorb any CPU time not being used to perform the required foreground operations. This is an important feature where fast response times are required, but where the overall real-time system load is small. The BCM makes use of the special Sigma 2/3 hardware for memory protection and priority interrupts to accomplish the concurrent foreground/background operation.

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

Many input/output editing features are available to speed paper tape operations. Through a unique system of software "command chaining" the BCM is able to drive low-speed, byte-oriented peripherals at full speed without interfering with real-time responsiveness in the system.

The BCM provides two levels of logical (rather than physical) device referencing, enabling system configurations to change or grow without reprogramming. Further, through many device-independent features and by the use of standard media formats, input and output can be directed to card equipment, paper tape equipment, or magnetic tape with no changes in the user's program.

By use of a special system initialization program, each installation can specify the peripheral devices available, the hardware options and interrupt levels, and the amount of core memory to be devoted to both real-time foreground and background.

The Symbol assembler, Basic FORTRAN compiler, Concordance program, mathematics library, and paper-tape utility routines are available under the BCM. Limited debugging capability is also available.

HARDWARE CONFIGURATION REQUIREMENTS

The minimum hardware configuration requirements are

1.	Sigma 2 CPU or Sigma 3 CPU	SDS Model 8001 or SDS Model 8101
2.	Core memory (4096 words)	SDS Model 8051
3.	Memory parity[†] interrupt (includes watchdog timer in Sigma 3)	SDS Model 8012
4.	Memory protection	SDS Model 8014
5.	One interrupt level (for BCM Control Task)	SDS Model 8022, 8023, or 8011 (external, integral, or counter-equals-zero)
6.	Memory increment (4096 words)	SDS Model 8053
7.	Keyboard/Printer with paper tape reader/punch	SDS Model 8092

[†]Hereinafter referred to as "the Monitor" or "the BCM".

[††]A CPU miscrosecond is defined as the amount of CPU time used when no I/O is in progress. The figure of 100 assumes multiply/divide hardware. Multiply/divide software simulation takes about 250 microseconds at the multiply/divide interrupt level.

[†]Memory parity and memory protection are required only for concurrent foreground/background.

A Keyboard/Printer (SDS Model 8091) and a Paper Tape Input/Output System (SDS Model 7060) are a highly recommended substitution for the SDS Model 8092.

An additional interrupt level is required for each foreground task under the BCM.

BCM SUBSYSTEMS

A variety of subsystems and processing programs are available under the BCM, all of which operate as background programs within the system.

LANGUAGE TRANSLATORS

SYMBOL

The basic Symbol assembler provides the user with a symbolic, machine-oriented language and a language processor. The assembler accepts a program coded in Symbol language, processes it, and outputs a binary object program and an assembly listing. The standard object language format is described in Appendix A.

The Symbol assembler and Concordance program are fully described in the Sigma 2/3 Symbol Reference Manual (Publication No. 90 10 51).

BASIC FORTRAN

Basic FORTRAN is a mathematically oriented language and processor that provides simplified programming for scientific, engineering, and mathematical applications. Basic FORTRAN is completely described in the Sigma 2/3 Basic FORTRAN Reference Manual (Publication No. 90 09 67), and Sigma 2/3 Basic FORTRAN Operations Manual (Publication No. 90 10 61).

SERVICE PROGRAMS

CONCORDANCE

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

The omission of the optional control cards yields a standard Concordance listing containing all program symbols except standard operation and directive code mnemonics.

LINKING LOADER

The Linking (relocatable) Loader performs the following functions:

1. Loads one or more binary object programs from the BI (binary input) device.
2. Resolves all cross references among programs and subprograms, and processes all calls to library routines.
3. Writes a memory map on the LO (listing output) device, showing the address of each external definition in the object program.
4. Updates instructions in the loaded program with corrections supplied at load time.

SYSTEM LOADER

The System Loader is an extended version of the Linking Loader. It allows for the preparation of foreground tasks, background programs, and processors from absolute or relocatable decks. A complete description of the System Loader is given in Chapter 5.

UTILITY SUBSYSTEM

The Utility processor operates in the background and provides the BCM user with a media copy routine, a record editor, an object module editor, a dump routine, and a sequence number editor.

A complete description of the Utility Subsystem is given in Chapter 9.

DEBUG PROGRAM

The Debug program operates in the background and permits the BCM user to dump selected portions of memory in a hexadecimal format. This is a highly desirable feature to expedite debugging. Debug can be loaded like any library routine. A description of the Debug program is given in Chapter 10.

BASIC DEFINITIONS

TASK

A task is an entire set of operations performed independently of other operations in the system. A task, logically, consists of three parts (that may or may not be physically contiguous).

1. A Task Control Block (TCB) that contains both status information and the contents of the registers from the interrupted task (see Table 13).
2. A task body that consists of a sequence of instructions executed in response to the task interrupt.
3. A task temporary storage area for the Monitor service routines that provides reentrancy for these routines.

Examples of tasks are:

1. Real-time foreground routines connected to external interrupts.
2. Monitor I/O Interrupt routine.
3. Monitor Control Panel Interrupt routine.
4. Each of the override group of interrupts.

5. BCM Control routine (for loading, abort, etc.).

6. Background program, that operates as a single task.

A task may use Monitor service routines (defined below) but must never "branch" to another task. One task may "trigger" another task at the interrupt level of the receiving task by means of a Write Direct instruction. There is a prescribed entrance and exit procedure for real-time tasks in the system, described in Chapter 7.

PROGRAM

A program is one or more tasks (and, optionally, some common data storage) that are loaded and controlled as a unit. There are two types of programs under the BCM:

1. Resident foreground programs that consist of one or more tasks, special routines for receiving I/O interrupt responses, and any common storage that may be needed.

2. Background programs, consisting of a single task.

FOREGROUND

Foreground refers to the real-time or Monitor tasks that are operated in protected memory on a real-time basis. There can be any number of foreground tasks, up to the number of internal and external interrupts possible in the system. However, since all foreground tasks must be resident, the fundamental limitation is the amount of core space available.

BACKGROUND

Background refers to a nonreal-time program executed in nonprotected memory, when such memory is available. The background program uses available CPU time (that is not needed by the real-time foreground tasks) to provide higher efficiency in the system. Background programs may be assemblies, compilations, or data processing programs. There are two fundamental restrictions in background programming:

1. The Sigma 2/3 hardware and the BCM software must completely and absolutely protect the resident foreground programs from the background program, in terms of I/O and core memory protection. Thus, an undebugged background program is never allowed to interfere with real-time foreground tasks; it must operate in a nonprotected memory, and it must use the Monitor service routines for all I/O or other privileged operations.

2. The background program must use the CPU time that is available after the real-time foreground is satisfied. Thus, the background program will not be guaranteed any processing time if the foreground is very active. The background must not inhibit interrupts or do anything else that would interfere with real-time foreground responsiveness.

MONITOR SERVICE ROUTINES

These are resident parts of the BCM that can be used by real-time foreground tasks, by the background task, or by BCM tasks. The routines are all coded in a reentrant manner and those that require temporary storage use the temporary stack space pointed to by the TCB for each task.

PRIORITY LEVEL

The Interrupt Priority Sequence (described in detail in the SDS Sigma 2 and Sigma 3 Computer Reference Manuals), is the basis for the priority level of tasks in the BCM system. That is, the priority level of a task is dependent on the position of the associated hardware interrupt in the interrupt priority chain. Thus, no two tasks in the system have the same priority level. The background program is not connected to any priority in the system, i.e., below any of the hardware priority levels.

TEMPORARY STACK

This is a block of core storage associated with a particular task, and is used by the Monitor service routines for temporary storage to achieve reentrance in the service routines. An entry in the TCB for a task points to the temporary stack space. When the task is active and is using either the Monitor service routines or the floating accumulator (defined below) the beginning of the temporary stack space for the active task must be set into core memory location 0006 (after the previous contents of 0006 are saved).

When the Monitor service routines need temporary space, they load the contents of location 0006 into the base register, and use this to point to the temporary stack space for the active task. The Monitor service routine M:SAVE sets this pointer. The background temp stack is the first 32 words of background space.

FLOATING ACCUMULATOR

This is a software convention that is used extensively by the FORTRAN compiler, the mathematics library, and also can be used by any Symbol programs. The floating-point accumulator is assumed to occupy the first five locations of the temporary stack space for each task that uses a temporary stack space. The floating-point accumulator is used like a hardware accumulator, i.e., to build up a cumulative result from single-precision real (floating-point) calculations. The single-precision real number is assumed to occupy the first two of the five locations. The third and fourth locations are used for temporary scratch storage, and the fifth location contains error flags.

As a convenience in referencing the floating accumulator, locations 0001 through 0005 are set with pointers to the actual core locations. This is done when entry is made to the active task (and is performed by the Monitor service routine M:SAVE when the routine is used). Location 0001 contains the value of location 0006, location 0002 contains the value in location 0006 plus 1, and so forth. Therefore, indirect addressing on locations 0001 through 0005 will result in storing, loading, or modifying the actual floating accumulator.

BCM CONTROL TASK

The BCM Control Task controls the reading of control commands, loading background programs, interpreting unsolicited key-ins, and aborting or terminating the background job. The BCM Control Task must be connected to the lowest priority hardware interrupt in the priority sequence during system initialization.

The BCM Control Task uses the same entrance and exit procedure and the same type of TCB that a real-time foreground task does. Since its main function is to control the background space, it must be lower in the priority sequence than the real-time tasks.

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 for purposes of unsolicited key-ins. Also, if the background task is in a loop and must be aborted, the Monitor must have a hardware priority level available for immediate control.

All of the BCM functions associated with this level operate as subtasks to the BCM Control Task and are not reentrant.

DEVICE-FILE NUMBER

The device-file number is a logical means of referring to a physical, peripheral device. The term includes both the words "device" and "file" to imply both a physical device and a collection of information on the device, since the current position of the device is associated with the current file of information.

The device-file number is an integer, and the set of all valid device-file numbers in the BCM system is the set of integers from 1 to n (defined at system initialization).

The device-file number is an index to a table of information, maintained by the BCM, that concerns the activity associated with both a particular device and a current file on the device. (The use of the device-file number is explained more fully in Chapter 8.)

OPERATIONAL LABEL

The convention of operational labels is used for the processors (such as the Symbol assembler or the Basic FORTRAN compiler) to make them device-independent, and is also used to give some mnemonic value to the input/output operations associated with the processors.

An operational label is a two-byte EBCDIC name that is used as a label in referring to a device-file number.

The standard operational labels can be reassigned to different device file numbers by means of system initialization changes or by an ASSIGN control command. There is one table of operational labels used for the background and another for the foreground. (The device file numbers are also stored as binary integer values in the two tables that correspond to foreground and background use, respectively.)

DEVICE NUMBER

The device number is a two-character hexadecimal representation of the physical device number shown in the device selection switches. It is generally set when the system is installed, and need not be changed unless the device selection switches are changed.

DEVICE TYPE

A device type is a name (and a collection of characteristics) associated with a particular class of peripheral devices. There can be either one device on the system belonging to a given device type, such as a CR device (for a card reader), or there can be several devices of the same type, such as MT (for magnetic tapes). This is a convenient method of referring to a device and an economical way to contain information common to several devices.

DEVICE UNIT NUMBER

Basic FORTRAN refers to peripheral devices by an integer value, called a device unit number. The device unit numbers can be equated to a device-file number by FASSIGN control commands as a way of equating the device unit numbers to an actual peripheral device, or can be defined at systems generation.

BCM CHARACTERISTICS

RESIDENT SECTION

The Basic Control Monitor consists of the following resident parts:

1. Several independent tasks (memory parity, Multiply/Divide, etc.) that operate from the hardware interrupts, as do the real-time tasks. The tasks are not reentrant. They may communicate with one another and may use some of the Monitor service routines, as do real-time tasks.
2. Several reentrant Monitor service routines, used by any tasks in the system. These are described in Chapter 6.
3. Constants and tables in the zero-table (locations 0 to 255, decimal). See Core Memory Allocation below for a description of the zero-table.
4. Input/output constants and status information, outside the zero-table.

NON-RESIDENT SECTION

The non-resident part of the Basic Control Monitor (system initialization portion) is loaded into core storage from 8K downward, and is used to select the optional features of the BCM and also to initialize the input/output constants.

PRIORITY LEVELS

The relative priorities of the separate Monitor tasks are shown in Figure 1. Although these tasks are not reentrant

(there is no need for them to be reentrant), they are serially reusable; that is, as soon as one of these tasks finishes processing a request for one operation or real-time task, it can then immediately process another operation or request. For example, I/O interrupts are processed one at a time, with the highest priority device always being processed first if several interrupts are waiting; but as soon as one interrupt request is completely processed, another request for a separate device can then be processed.

Highest	Memory Parity Error
	Protection Violation
	Multiply Exception
	Divide Exception
	Real-time Task(s), if any higher than I/O
	Input/Output
	Control Panel Interrupt
	Real-time Task(s), if any lower than I/O
	BCM Control Task (lowest hardware level)
Lowest	Background (lower than all hardware levels)

Figure 1. Relative Priority Levels

The guiding philosophy of the Monitor tasks is to keep the processing of the background below the priority level of the real-time foreground operations (which is why the special interrupt level is required for the BCM Control Task), and to keep all processing at the higher Monitor task levels (such as the I/O task level or Control Panel interrupt level) as brief as possible. A short description of each of the BCM tasks follows below.

MONITOR TASKS

MEMORY PARITY

This task is responsible for examining memory parity errors. If a memory parity error occurs for the background program, the background program is aborted and the real-time foreground is not disturbed. The Memory Parity Task calls the reentrant Monitor routine M:ABORT but does not actually do the aborting at this level. The routine M:ABORT sets an abort flag and triggers an interrupt at the BCM Control Task level that actually aborts the background and prints an error message. The BCM will halt and display the bad address in the A-register if the parity error is in protected memory.

③ In a Sigma 3 configuration, the interrupt associated with the memory parity error is also related to one of two types of watchdog timer runouts. The first type of timer runout is caused by an attempt to reference an unrecognized device during direct I/O, and the second type is caused by an unexplained delay during an integral IOP call. If a timer runout occurs because an unrecognized device has been referenced during direct I/O, the Memory Parity Task will test for a watchdog timeout receiver.† If no receiver is specified, the task will then trigger another task (at the BCM priority level) to write an appropriate message. The BCM will then continue with the foreground task. If the second type of timer runout occurs, the Memory Parity Task takes the same action as for a foreground parity error and the BCM will halt, except that in this case the Overflow Indicator will be set. An integral IOP timer runout indicates hardware problems.

PROTECTION VIOLATION

Any attempt by the background to modify the contents of protected memory or to execute a privileged instruction will cause the Protection Violation interrupt to abort the background program, using the same method as the Memory Parity interrupt, above.

MULTIPLY/DIVIDE EXCEPTION

This task simulates and subsequently executes a Multiply or Divide instruction for Sigma 2/3 computers not equipped with Multiply/Divide hardware. The task is not reentrant, so all lower-level interrupts are locked out for the duration of the simulation (approximately 250 to 300 CPU microseconds).

INPUT/OUTPUT

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 and 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 an AIO receiver was specified (see Chapter 8 on Input/Output for a description of an AIO Receiver), control is transferred to this receiver at the I/O priority level. It is expected that the AIO Receiver will return to the Input/Output Task so that the task can exit properly.

CONTROL PANEL

A Control Panel Interrupt causes the Control Panel Task to set a flag for the BCM Control Task, trigger the task, and then exit from the Control Panel Task in about 40 to 50 microseconds of CPU time. The operator response is processed at the level of the BCM Control Task.

BCM CONTROL

This task controls the background operations. It is the only BCM task that actually performs input/output and, therefore, is the only task that requires temporary stack space for the reentrant BCM input/output routines.

---

†A watchdog timeout receiver is specified by loading a permanent task into foreground. This task must have an initialization routine that stores the address of the receiver in the first cell preceding the program status doubleword (PSD) of the Memory Parity Task. A pointer to the PSD is located in cell X'102'. ③

CORE MEMORY ALLOCATION

The following rules must be followed in regard to core memory allocation:

1. Background space must be in the upper part of available memory and must begin on a page boundary (a page is 256 words on 256-word boundaries). It is necessary to allocate at least one page of background.
2. The first 256 words of lower memory are reserved for constants and user communication region.
3. The region from 256 (decimal) to 399 is reserved for internal and external interrupt levels; if all the space is not required, the Monitor uses the remainder of this region for table space.
4. The resident BCM begins loading at 400 (decimal) and continues upward in lower core. Only the required Monitor options are actually loaded.
5. Each resident foreground task must be explicitly and directly connected to its proper interrupt level.
6. All of memory, except the background space, is protected (on a multiple of 256 words). An example of core memory allocation is given in Figure 2.

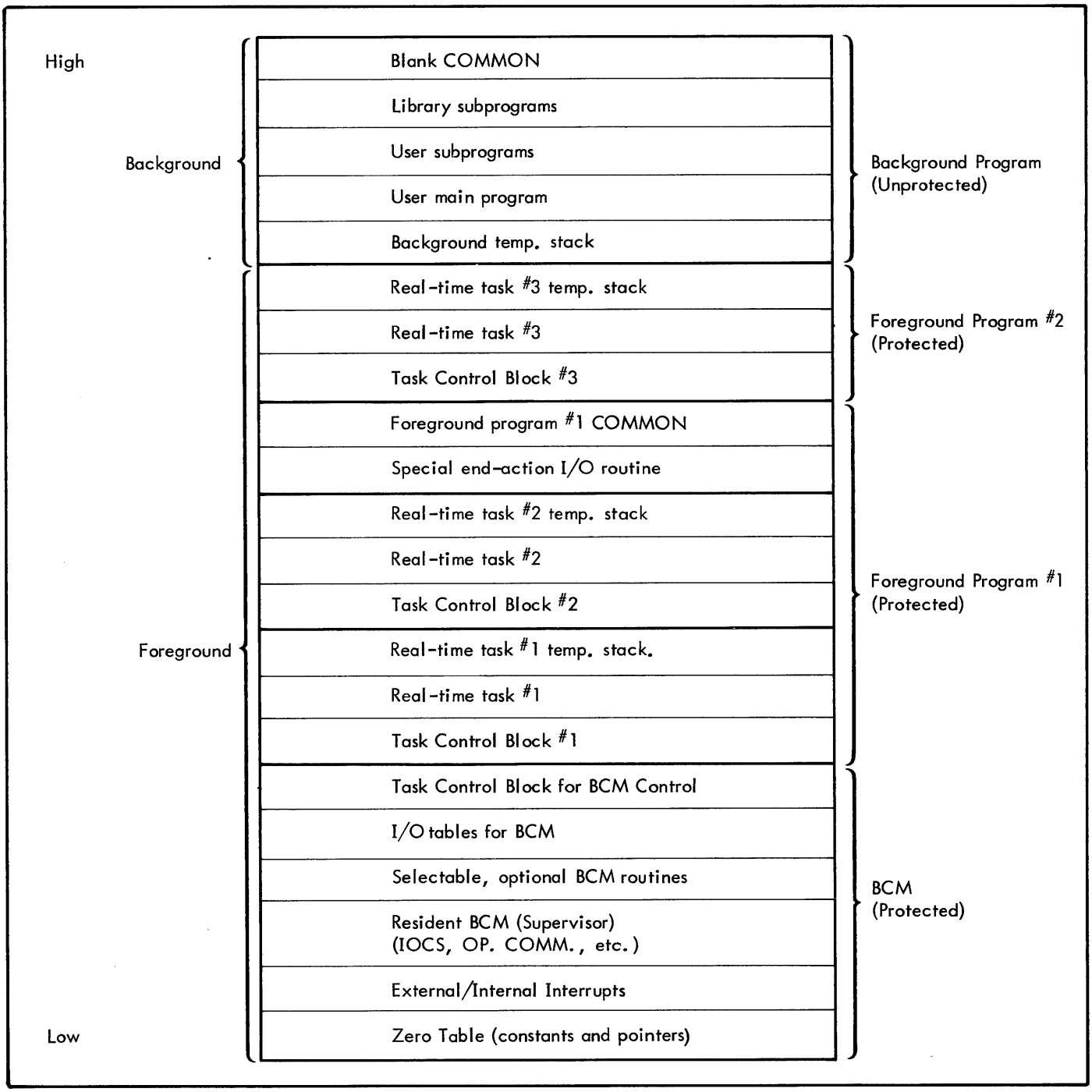


Figure 2. BCM Core Memory Allocation (Example)

2. CONTROL COMMANDS

INTRODUCTION

The Monitor is controlled and directed by means of control commands. These commands effect the construction and execution of programs and provide communication between a program and its environment. The environment includes the Monitor, SDS Basic FORTRAN, Symbol assembler or other processors, the operator, and the peripheral equipment.

Control commands have the general form

```
! mnemonic specification
```

where

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

mnemonic is the mnemonic code name of a control function or the name of a processor. It must begin immediately following the ! character. Only the first four columns are used to identify the command.

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 integer values. Specification fields are separated by a comma.

In this manual, options that can be included in the specification field of a given type of control command are shown enclosed in brackets (although brackets are not actually used in control commands).

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. Comments detailing the specific purpose of a command may be written following the command terminator, but no control command record may contain more than 72 characters.

Communication between the operator and the Monitor is accomplished via control commands, key-ins, and messages. Control commands from the CC device are usually input to the Monitor via punched cards; however, any input device(s) can be designated for this function (see "ASSIGN", below).

Control key-ins are always input through the typewriter. All control commands and Monitor messages are listed on the output device designated as the listing log (normally a line printer). In this manner, the Monitor keeps the operator informed regarding the progress of a job. A summary of the control commands is shown in Table 1.

Monitor service control commands follow.

ABS Each absolute binary program to be loaded into core memory is preceded by an ABS control command. The binary program that follows must be in Sigma 2/3 standard object language format (see Appendix A), and must not contain any SREFs, REFs, or DEFs. The deck may be a user's program for the background, a processor for the background, or a real-time program for the foreground. The binary program is read from the BI device.

The form of the ABS control command is

```
!ABS name
```

where

name is the two-character name of the program or task that follows. If the name is for the processor that is to follow, it can be three characters or more. However, the first three characters must be identical to the first three characters appearing in the subsequent processor control command.

This command is not free field in format. The name must start in column 6. The binary deck can be produced from an assembly, or can be output from the System Loader. The transfer address in the end module is used as the entry point into the program. The deck must contain the name in the start module. This name is created by the FORTRAN compiler for a main program, by the IDNT directive in Symbol, or by the ID command in the System Loader. A foreground module must be preceded by an FG key-in.

ASSIGN The ASSIGN control command causes a new operational label to be equated to a specified file number, or causes a standard operational label to be redefined. Operational labels are reset to the standard values at the beginning of a job by the BCM Control Card Interpreter. When a standard assignment is redefined, it remains in effect only until a JOB command is encountered or until it is again redefined. If an assignment is made to file zero, it indicates that the label is not effective, and all references to this operational label result in a no-operation until it is redefined.

ASSIGN commands may appear anywhere within the control command stack, and take effect immediately. That is, if an ASSIGN command redefines the CC device, the very next control command is read from the newly defined device. ASSIGN commands can define or redefine both foreground and background operational labels.

The form of the ASSIGN command is

```
!ASSIGN operational label = file number [,F]
```

where

operational label — is one of the two-character alphanumeric names in the foreground or background operational label table (or is to be placed in the table).

file number — is the device-file number for some physical device in the system.

F — when present, declares that the file number is to be included in the foreground operational label table; otherwise, it is assumed to be in the background operational label table.

C: The C: control command causes the specified real-time foreground task to be connected to a specified interrupt location and optionally armed and enabled (as the control code specifies); it can also be triggered by a second connect operation if the code is equal to 7. (See "Task Control Block Functions" in Chapter 7.)

The form of C: control command is

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

where

tcb — is the address of the Task Control Block for this task. As noted above, if the value is hexadecimal, it must be shown as +xxxx.

code — when present, overrides the initial interrupt function control code in the TCB for the task; a code of 7 would cause the level to be triggered. The code in the TCB is not changed.

Warning: Use codes 0 and 6 with great caution. Code 0 is undefined, and code 6 will disable all other interrupts in the group.

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

The form of the EOD control command is

```
!EOD
```

FASSIGN This is identical to the ASSIGN control command, except that it operates on FORTRAN device unit numbers and not on operational labels.

The form of the FASSIGN control command is

```
!FASSIGN device unit number=device-file number [,F]
```

where

device unit number — is an integer in the foreground or background operational label tables (or is to be placed in the table).

device-file number — is the device-file number for some physical device in the system.

F — when present, declares that the file number is to be included in the foreground operational label table; otherwise, it is assumed to be in the background operational label table. The F specification on FASSIGN must be preceded by an FG key-in.

FIN The FIN control command is used to specify the end of a stack of jobs. When the FIN control command is encountered, the Monitor outputs the command on the listing log to inform the operator that all current jobs have been completed, and then enters the idle state.

The form of the FIN control command is

```
!FIN
```

FSKIP The skip file control command (FSKIP) causes a specified magnetic tape to be spaced forward, either immediately past the next tape mark, or past the nth tape mark if n files are specified.

The form of the FSKIP control command is

```
!FSKIP device [,number]
```

where

device — is the device-file number of the tape to be spaced forward. It is restricted to magnetic tapes for background usage.

number — is the number of files to be skipped; if absent, one file is skipped.

JOB The JOB control command is used to signal the beginning of a new job. The background operational label table is reset to the standard device-file numbers defined at system initialization, as are the FORTRAN device unit numbers. Any device-file numbers for the background with

pending input/output are reset, and the background core memory is reset to all zeros. The use of this command is optional.

The form of the JOB control command is

```
!JOB
```

PAUSE The PAUSE control command is used to temporarily suspend the background program loading operation. It can be used to indicate the end of a reel of paper tape, thus allowing the operator time to change reels. When the operator performs an unsolicited "S" key-in, the Monitor reads the next control command. This command is listed on the OC device.

The form of the PAUSE control command is

```
!PAUSE [comments]
```

REWIND The REWIND control command is used to rewind a magnetic tape. It has no effect on the other devices. The operation takes place immediately when the command is interpreted.

The form of the REWIND control command is

```
!REWIND device
```

where

device is the device-file number of the tape to be rewound. It is restricted to background files.

UNLOAD The manual rewind (UNLOAD) control command causes the specified device to be rewound in the manual mode, so that operator intervention is required to use the device again.

The form of the UNLOAD control command is

```
!UNLOAD device
```

where

device is the device-file number of the tape to be unloaded. It is restricted to background files.

WEOF The write end-of-file (WEOF) command causes an end-of-file mark to be written on the output device, if it is appropriate to the device. For magnetic tape, a tape mark is written. For the card punch or paper tape punch, an EOD command is written.

The format of the WEOF control command is

```
!WEOF device
```

where

device is the device-file number of the device that is to have an end-of-file written on it. It is restricted to background files.

The valid BCM control commands are listed in Table 1.

Table 1. BCM Control Commands

BCM Service Commands
ABS ASSIGN C: EOD FASSIGN FIN FSKIP JOB PAUSE REWIND UNLOAD WEOF
BCM Processor Subsystem Commands
BFORTRAN CONCORDANCE LOAD SYMBOL UTILITY SLOAD

PROCESSOR SUBSYSTEM CONTROL COMMANDS AND BCM INTERFACE

The Monitor operates independently of secondary storage for the processors and will not load the processors from secondary storage even if it is available.

The details of the control commands for each of the standard processor subsystems are defined later in chapters for the individual subsystems. However, there are some rules common to all of them, as follows:

1. All of the processors operate in the background space.

2. The processors use the standard background operational label table assignments for their I/O requests, with the exception of the UTILITY program. (See Table 7 for the standard background operational labels.)

3. The first character of each line of the listing output from the processor is always interpreted as a vertical

format character (a carriage control) and is never printed. The BCM I/O routines treat the vertical format properly for keyboard/printer, line printer, and magnetic tape.

4. When the BCM transfers control to each processor, the X register contains the address of the control image.

5. All processors use the standard system constants, as defined in Table 2, where applicable.

6. 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 record. The M:CTRL routine will treat these operations properly for the devices involved, 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 also writes a blank card image on the BO device if that device was used for the operation.

CONTROL COMMANDS IN MINIMUM BCM SYSTEMS

In a minimum BCM configuration, without the full control command interpreter (CCI), only ABS, EOD, and FIN are recognized as valid service commands. The processor subsystem commands are all recognized, however.

SAMPLE DECK SETUPS UNDER BCM

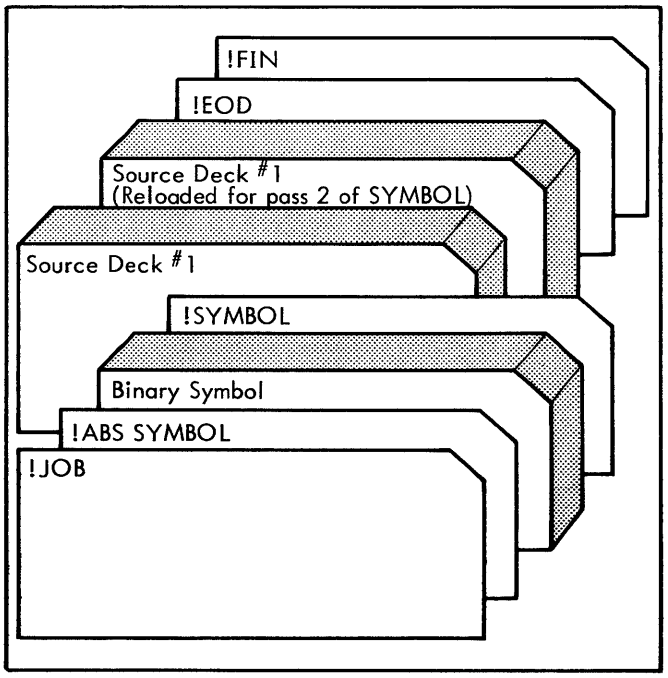


Figure 3. Assembly without Magnetic Scratch Tape

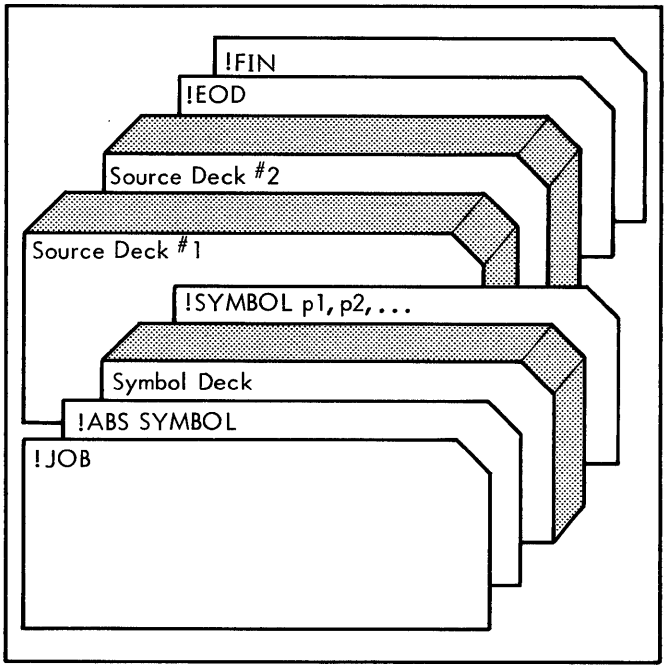


Figure 4. Assembly with Magnetic Scratch Tape

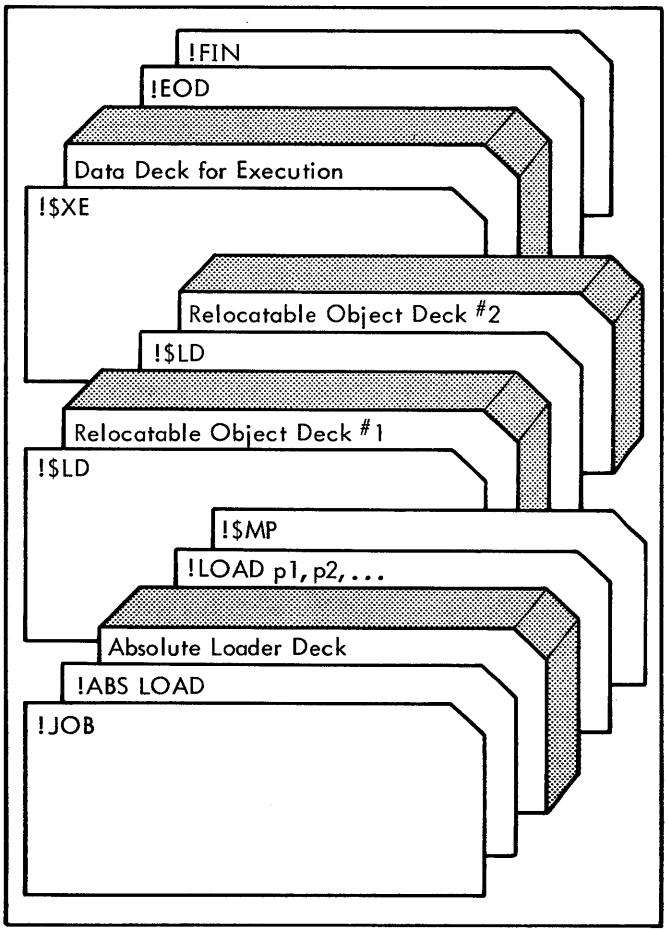


Figure 5. Load Example for Background Program

3. OPERATOR COMMUNICATION

SYSTEM COMMUNICATION

Occasionally, the Monitor may inform the operator of unusual conditions in the BCM system via Monitor typeouts. These will be directed to the operator's console through a device file that is assigned to the Keyboard/Printer.

If the message is concerned with some I/O error condition, the Monitor I/O routine that generated the message will be waiting to sense a change-of-state in the device from automatic to manual and back to automatic; or from manual to automatic, if the device was in the manual mode after the operation was attempted. When this change-of-state is sensed, the operation is retried.

MONITOR TYPEOUTS

The possible Monitor typeouts under the BCM are

!!KEY ERROR The Monitor did not recognize a key-in; a new key-in should be initiated.

!!ABORT CODE xx LOC yyyy The background job has been aborted by reason of code xx at location yyyy. The standard Monitor abort codes are given in Appendix B, but any two EBCDIC character codes can be given by the routine that called for the abort. The routine sets the index register to the two EBCDIC characters of the abort code on entry to M:ABORT and sets the A register to the location value.

!!MACHINE FAULT AT LOC yyyy An attempt has been made to reference an unrecognized device during direct I/O. The message will not appear until the BCM Control Task becomes active.

!!CCI Begin reading control commands from the CC device. This is the indication that the previous job has terminated normally and that the BCM is ready for the next background job. This typeout occurs after an S key-in when no background job is active.

!!PAUSE comments A PAUSE control command has been read. The comments field may contain messages or instructions to the operator. A control panel interrupt and a key-in of 'S' will cause the BCM to continue reading from the job stack.

!!dtnn FAULT Some condition on device type dt with physical device number nn (hexadecimal) has put the device in a nonoperational status. The recovery procedure is described above (in the discussion under change-of-state). The operation is automatically retried when the device is returned to the automatic mode. It is not necessary (or possible) for the operator to type in a response.

!!dtnn ERROR There is a parity, transmission, or data rate overrun error on the device. Any specified retries have been performed before the message is output.

!!dtnn PUNCHES An invalid punch combination has been sensed on the EBCDIC card.

!!dtnn EMPTY The device specified is in the manual mode and may be out of paper, cards, or tape.

Real-time programs with special requirements can inform the operator of special conditions and wait for an operator response; however, the error messages are primarily designed for background programs.

The ATTENTION switch on magnetic tape units is for special use under the RBM (Real-Time Batch Monitor) and has no effect in BCM error recovery.

OPERATOR CONTROL

UNSOLICITED KEY-INS

Because of the possible time delays associated with messages to the operator and subsequent responses, no devices used for an operation with a critical time factor should time-share an I/O channel used for operator communication.

All background references to the operator output device should be made to the OC operational label. One method of operator control is in answer to a specific request from a foreground or background program. In this case, there is no standard format.

However, the operator may also desire to exercise control over the background programs on an unsolicited basis. This control (unsolicited key-ins) is initiated by the operator placing the INTERRUPT switch on the Sigma 2/3 Processor Control Panels at the INTERRUPT position. This causes an interrupt into the Control Panel Task. The task sets a BCM Control Task status flag, issues a Write Direct to trigger the BCM Control Task, and then exits.

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

!!KEY-IN

and then requests a two-character input from the operator. Each response must be terminated with the NEW LINE code. The backspace (¢) and delete (EOM) codes may be used before the NEW LINE is typed, to correct the key-in. The analysis and subsequent action from this unsolicited key-in are performed at the BCM Control Task priority level. If there is a second control panel interrupt before the first one is processed, the second one is ignored.

The specific responses possible under BCM are given below.

S Continue processing, if the Monitor is in an idle state (the Monitor can be put into an idle state from a "W" key-in or a PAUSE or FIN control command). If there is an active background program, continue processing it. If there is no active background program, begin reading control cards from the CC device.

W Either the Background or BCM Control Task will go into an idle state, whichever one is active (foreground tasks are not affected).

X Abort the background job without dumps but with the error code OP and a printed message that shows the location of the last background instruction executed.

KP Reassign CC to the keyboard/printer. This is useful if CC has been improperly assigned.

FG For any operation that intends to modify the foreground (such as ABS, ASSIGN, C:, or FASSIGN), it is necessary to first perform a key-in of FG or this foreground modification will not be permitted and the intended operation will be aborted. The key-in is effective until the next FIN command. It is merely intended as an additional check for foreground modification.

CP After a JAM A or a JAM B (or other unusual condition) on the card punch that results in the card punch stopping, clear the jam, place the punch in START, and input the CP key-in. This will cause the previous (erroneous) card to be repunched. !!CPnn FAULT will be typed on a keyboard/printer, device-file number 1, and the card punch will then continue punching in a normal manner.

4. LINKING LOADER

INTRODUCTION

The Linking Loader used with the Sigma 2/3 Basic Control Monitor reads binary modules in Standard Object Language format (see Appendix A), loads them into background memory, and links modules with external references and definitions. One or more libraries can be selectively loaded to satisfy primary and secondary external references. Load map, program modification, and execution capabilities are optional.

The Monitor transfers control to the Linking Loader upon reading a LOAD control command. After initialization of the Linking Loader process in the background, the loader reads subcommands preceded by the special characters !$.

OPERATING SEQUENCE AND OPTIONS

After the Linking Loader is loaded by the Monitor Absolute Loader, the Linking Loader moves itself to the top of available memory, clears the remainder of the background to zero, and initializes itself. It then sets a symbol table origin in a manner that will permit the symbol tables to build from the origin of the Linking Loader to the beginning of the background.

The load/execution origin is initialized to the value of K:BACKBG (beginning of background) plus X'20'. The X'20' locations are reserved as temporary storage for the Monitor service routines used by the background program. The general memory areas involved are shown in Figure 6.

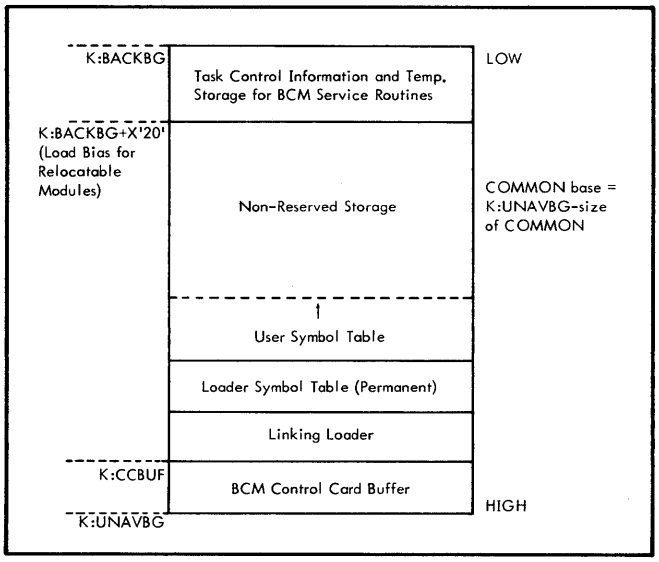


Figure 6. Background Memory Allocation

The following steps then take place:

1. Binary object modules are loaded, beginning from this origin.
2. Checksum, record sequence number, item type, error severity, and control card sequence are checked.
3. The last transfer address encountered is saved as the execution address (see $Xm in this chapter).
4. COMMON size is allocated by taking the larger value of the 'csize' parameter in the LOAD control command, or the first nonzero common size located in the START item of a binary module.
5. Optionally, a concurrent map of module names, beginning addresses, external definitions and their values, and unused or doubly-defined external definitions are output on DO.
6. All loader control commands read are listed on DO.

To satisfy external references whose values were not defined during the main loading, one or more libraries can be loaded, and modifications to either absolute locations or locations relative to external definitions can be made.

Program execution that is dependent on a severity level can be initiated, with a choice of either a transfer address specified on a !$XZ or !$XR control command, or a transfer address encountered during loading.

If the asize option is exercised, it allows an absolute FORTRAN Run-Time package to effectively use up the space occupied by the Linking Loader and the symbol table. This facility is performed by having the Linking Loader overlay itself with the program it just loaded. The Absolute Run-Time package is then read into the beginning of background to occupy the space vacated by the program. The parameter "asize" is determined by the size of the Run-Time package.

References to Monitor service routines are satisfied by definitions in the permanent symbol table.

LOADER CONTROL COMMANDS

LOAD The LOAD command, read by the BCM from the CC device, causes the initiation of the Linking Loader process in the background. The loader will read subcommands (given below) to carry out the load, map, modify, and execute functions. The LOAD command has the form

```
!LOAD [csize] [,asize]
```

where

csize denotes an alternative COMMON allocation value. If missing, it is assumed to be zero. The first nonzero COMMON allocation located in the START item of a binary module will be compared with the value of csize, and the larger of the two values will be used for COMMON allocation. If no COMMON relocation item is encountered in a START item, the value of csize will be used. If used, COMMON resides at the highest core area available.

asize denotes that an Absolute Run-Time (L:A) is to be loaded at the beginning of the background. After loading is complete (!EOD), the loader moves the program just loaded to this increment above background (overlaying the Linking Loader), and thus creates room for the Absolute Run-Time. The Absolute Run-Time will be loaded using the !ABS command of the BCM.

If asize is nonzero, EBIAS (execution bias) is set to K:BACKBG+X'20'+asize, and BIAS (load bias) will be set to K:BACKBG+X'20'.

Upon encountering an EOD, the Linking Loader enters the hash code (BCM identifier) and the transfer address in the first two cells of background. It then moves the program to its execution location and returns control to the Monitor, to await the loading of the L:A Absolute Run-Time.

If asize is empty, the Linking Loader functions in a normal manner; i.e., loading and execution are to be at the same location, K:BACKBG+X'20'.

The following control commands are read from the CC device by the Linking Loader and are listed on DO after they are read. All Linking Loader mnemonics are preceded by the special characters !$. The general form of the command is

```
!$mnemonic  parameters
```

where

eight blank characters following the mnemonic terminate the control message. A single blank terminates the parameter string. (Thus, the card may contain comments.) The mnemonic and first parameter can be separated by 1 to 8 blanks. A comma must separate parameters.

$LD The $LD command causes the Linking Loader to load a single binary module in Standard Binary Format from BI and save the program name and transfer address, if any. A $LD command must precede each binary module to be loaded. The form of the command is

```
!$LD  bound
```

where

bound is an optional parameter specifying that the load bias is to be rounded to the next higher multiple of the bound value before loading this module (bound must be a power of 2). An empty bound field resets the bound to zero. Bound must be of the form

+value
label
label±value

where

label is an external definition

value is a hexadecimal number

The Linking Loader initializes the load bias to the value in K:BACKBG plus X'20'. The bias for loading the next module is calculated by adding the relocatable program length (in the END item) of the current program to its load bias. The result may be modified by the bound value.

$LB The $LB command causes selective loading to proceed from the LI medium only if there are unsatisfied primary references in the loader symbol table at the time the $LB control message is read. If there are no unsatisfied references, the library is passed over, but no programs are loaded and the next control message is read. Unsatisfied references cause the Loader to read the Library, and to load those programs having external definitions that satisfy the references.

The selective loading process terminates when

1. An EOD from LI is encountered, in which case a list of unsatisfied references is output on DO and a new control command is read.
2. All references are satisfied, in which case the remainder of the library is passed up to the next EOD and the next control command is then read.
3. An irrecoverable error occurs, in which case the loader aborts.

Note that the selective loading may produce additional external references that must be satisfied by definitions further on in either the same or another library. One library is scanned for each $LB control command. The form of the command is

```
!$LB  bound
```

where

bound is an optional parameter specifying that the load bias is to be rounded to the next higher multiple of the bound value before loading this module (bound must be a power of 2). An empty bound field resets the bound to zero. Bound must be of the form

+value
label
label±value

$MP, $ML The $MP and $ML commands cause a load map to be output on the DO device (see map formats later in this chapter). The map is written out as loading progresses. For both $MP and $ML, maps of main programs and subprograms (i.e., modules preceded by a $LD control command) are identical. For $ML, the name, starting location, external definitions, and transfer address of each module are

output on the map. $MP suppresses the external definitions in the library subroutines (loaded by $LB control command).

When used, a $MP or $ML control message must precede all other !$ control cards in the control card sequence.

The format of these control commands is

```
(!$MP)
(!$ML)
```

$MD The $MD control command indicates that modifications are to be inserted in specified locations in core storage. Modifications are of the form

```
!$MD  addr,mod1 [,mod2, . . . , modn]
```

where

addr is the memory location where the first "mod" is to be placed. Successive mods are placed in successive locations.

mod is the value to be stored in addr or a successive location. If a "mod" parameter is empty (denoted by successive commas) zeros are stored in the corresponding location.

addr and mod_i are of the form

+value
label
label±value

where

label is an external definition

value is a hexadecimal number

Examples:

!$MD +402C,+4C01,+41FC

!$MD LABL+2C,+4C01,LABL+1FC

Note: The "label" must have been defined prior to its use in a $MD card. A label must not be used if an asize has been stipulated.

$XZ, $XR The execution control commands, $XZ or $XR, initiate execution at the last transfer address encountered according to existing load severity levels. The general format of these commands is

```
(!$XZ)
(!$XR)   transfer address
```

where

XZ means execute if there are no load errors (severity = 0)[†]

XR means execute regardless of the number of load errors (severity ≥ 1)[†]

transfer address is an optional parameter that specifies where execution must begin. It supersedes any transfer address encountered in the loading process. It must have the same form as the "addr" parameter of the $MD command.

!EOD An !EOD may be substituted for a $XZ or $XR command in the control card stack. When the Loader encounters an !EOD in the proper context, it stores the name of the program just loaded and its entry address in the first two cells of background memory so that the program just loaded may be entered through BCM like a processor, by using a !"name" control card. In normal mode (non-asize mode) the name used on the Monitor control card to reenter must be the "idnt" of the last program with a transfer address in the load stack. If an asize has been stipulated, the Loader relocates the program and returns to the Monitor. The sequence: !ABS L:A, Absolute Run-Time, and !L:A will load the Run-Time and transfer control to the background program.

ABSOLUTE RUN-TIME JOB SETUP

The role of the Linking Loader in the loading and subsequent processing of a FORTRAN program that uses the Absolute Run-Time package and DEFs output by the System Loader (see Figure 9 in Chapter 5) is illustrated in Figure 7.

MAPPING

For each module preceded by an !$LD card, the map includes

1. the name of the module
2. the beginning address of the module
3. a list of external definitions that are suitably flagged, and the address of each definition
4. the value of any transfer address encountered in that module.

For either $MP or $ML, the map for each module loaded with a $LD commands includes items 1-4.

For both $MP and $ML, item 3 it output for external definitions in the permanent symbol table.

[†]See "Severity Levels" at the end of this chapter.

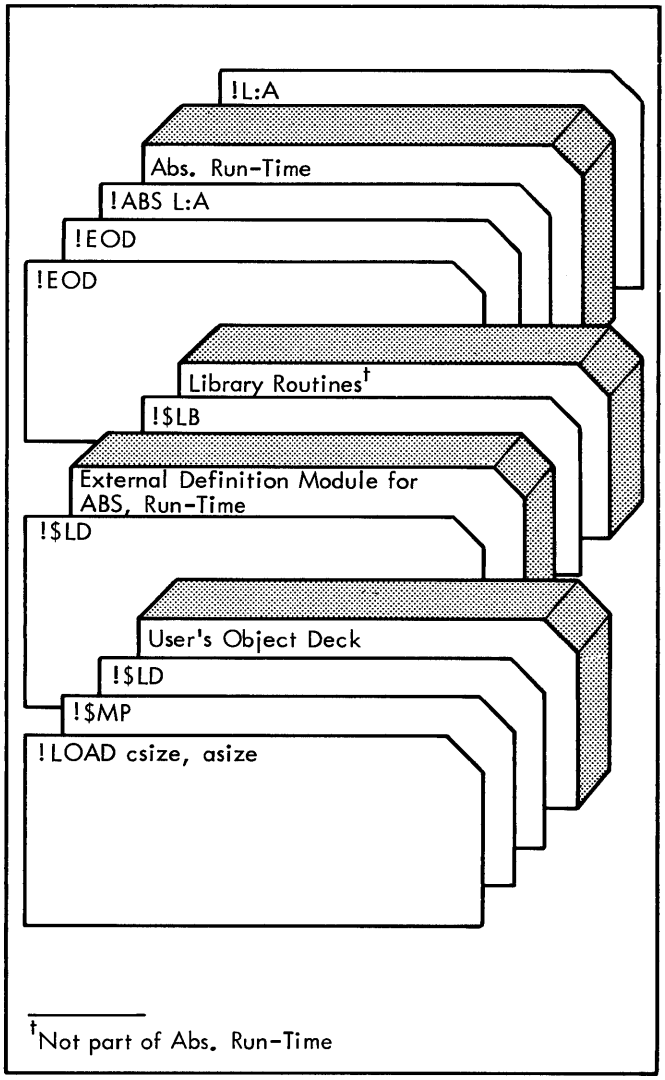


Figure 7. Overlaying Linking Loader for Absolute Run-Time

For $ML, items 1-4 are output for modules loaded with $LB commands; for $MP, only items 1, 2, and 4 are output for modules loaded with $LB commands. Even if no $MP or $ML Loader command is input, the listing on DO always includes the following.

1. Each loader command read.
2. Unsatisfied references at the conclusion of each library load or in response to an $XR or $XZ command before terminating the loader.
3. The COMMON storage base address.
4. The COMMON storage size.
5. The transfer address where execution will begin.
6. The error severity level.

The format of the map output for !$ML is illustrated in Figure 8.

```
!$ML
PERMANENT       definition      value
                definition      value
                    .               .
                    .               .
                definition      value

!$LD
PROGRAM         name            address
                definition      value
                definition      value
                    .               .
                    .               .
U               definition      value
D               definition      value
                    .               .
                    .               .
                definition      value
                                END TRANSFER value

!$LD
PROGRAM         name            address
                definition      value
                    .               .
                    .               .
                definition      value
                                END TRANSFER value
                    .
                    .

!$LB
LIBRARY         name            address
                definition      value
                    .               .
                    .               .
LIBRARY         name            address
                definition      value
                    .               .
                    .               .
                definition      value
LDERR  UR
                reference       chain
                    .               .
                    .               .
                reference       chain

!$LB
LIBRARY         name            address
                    .               .
                    .               .
LDERR  UR
                reference       chain
                    .
                    .
COM ADD value COM SIZ value ENDTRA value ERR SEV+n
```

Figure 8. Map Output Format

where

definition is the (up to) 8-character symbol for an external definition.

value is a 4-character hexadecimal address, number, or 'NONE'.

name is the program name or blanks obtained from the start module item.

address is the 4-character hexadecimal representation of the beginning load address for the program.

U means the definition is declared but not given a value.

D is a double definition.

END TRANSFER is the execution address from the END item of the module.

reference is the 8-character symbol for an external reference.

chain is the hexadecimal representation of the last address in the chain.

COM ADD is the base address for COMMON storage.

COM SIZ is the hexadecimal size of COMMON storage.

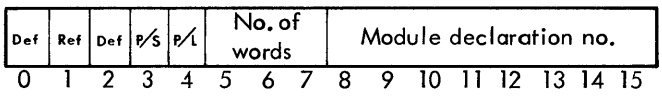
END TRA is the transfer address where execution begins.

ERR SEV is the error severity level (0 or 1).

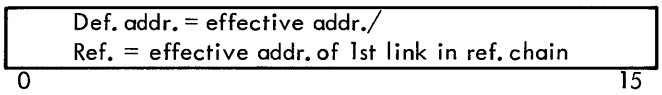
LOADER SYMBOL TABLE

The maximum allowable size of an entry is six words (up to eight characters) per symbol. The table is not ordered by the loader during the loading process. Entries are inserted as encountered. The entry size includes the control word.

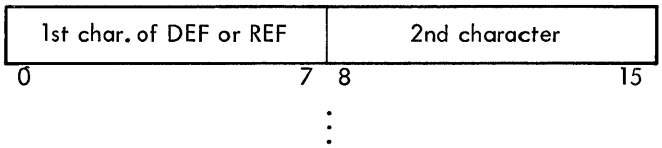
word 0



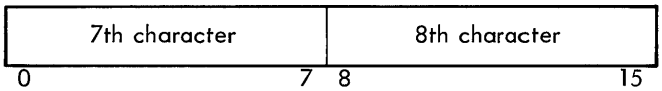
word 1



word 2



word 5



In this item, word 0 is the control word

where

Bit 0 is set only if the entry is a definition value. During loading, definitions are declared at the beginning of the module but are defined later in the module. The entry is made in the symbol table when the REF is encountered or the DEF is declared (see MAP). Bit 2 = 1 indicates that the entry is a definition declaration (the symbol has not been defined yet). Bit 0 = 1 indicates that a definition address or value has been inserted in word 1 of the table.

Bit 1 is set if the reference name is encountered prior to encountering a definition value. When bit 0 is set, bit 1 is reset. However, bits 1 and 2 may both be set at the same time if a reference and definition declaration are encountered before the definition value.

Bit 2 is set when a declaration is encountered. It is used for flagging (on the map) definitions that are declared but not defined.

Bit 3 defines an entry as a primary or secondary reference (assuming bit 1 is set). If bit 3 is set to a 1, it is a primary reference; if set to a 0, it is a secondary reference.

Bit 4 indicates whether or not the loader was in the library loading mode when the current symbol was encountered or defined. If bit 4 = 1, it designates loading of main programs and subroutines ($LD command). If bit 4 = 0, it indicates the library loading mode ($LB command).

Bits 5-7 indicate the length of an entry in words (the length can be 3 to 6 words). Trailing blanks in a symbol are suppressed.

Bits 8-15 are the declaration numbers of the entry. As a start item is encountered, the Loader assigns to the module a declaration number between 1 and X'FF' (this includes library programs). The number is used to locate the source of definitions for mapping (see MAP).

Word 1 is the effective address of the item. If the entry is a definition address, the effective address or value of the definition is contained in word 1. If the entry is a reference, the effective address of the first link in the threaded reference list (chain) is contained.

Words 2 to 5 contain the EBCDIC representation of the SREF, REF, or DEF. If the symbol entry contains less than 8 characters, trailing blank words are suppressed.

The symbol table is divided into two portions: the Permanent Symbol table, consisting of permanent definitions (principally for Monitor service routines, such as M:READ and M:WRITE); and the User's Symbol table, consisting of references and definitions encountered and entered during the load process. The permanent Symbol table may be altered only by reassembly of the Loader. Since the User's Symbol table is searched before the Permanent Symbol table, entries to the User's Symbol table supersede the definition in the Permanent Symbol table.

DIAGNOSTIC MESSAGES

Diagnostic messages are output on DO if errors are encountered during the loading process. For those errors considered irrecoverable, the loader first outputs the appropriate error message on OC and DO and then takes the abort exit to the BCM (see "BCM" for a description of abort procedures).

For recoverable loading errors, the error severity level is set to 1. Since space does not permit the listing of detailed information concerning the source of an error level of 1, the diagnostic messages are output on DO and thus may be interspersed with map information. The operator may abort the loading process by an unsolicited key-in at any time. The diagnostic messages are given in Table 2.

SEVERITY LEVELS

The possible severity levels for loading errors are as follows.

- 0 means no error severity on the module or no irrecoverable errors during loading (i.e., sequence number, checks, etc.).
- 1 means any nonabortable error, assembly/compilation errors, unsatisfied primary references, or no transfer address.

Table 2. Linking Loader Diagnostic Messages

Message	I. D.	Abort/ Continue	Explanation
LDERR	SQ	A	Sequence number on this record is incorrect.
LDERR	IB	A	A binary card has been encountered where an EBCDIC command card was expected.
LDERR	CS	A	The card cannot be checksummed correctly.
LDERR	IT	A	Illegal item type.
LDERR	TO	A	Symbol table overflow.
LDERR	IE	A	An EBCDIC card has been encountered before the end card of a module.
LDERR	CM	A	A COMMON displacement beyond the allotted COMMON size has been encountered. An error in COMMON allocation has occurred.
LDERR	CC	C	There is an error in the control command just read. Read another command from CC.
LDERR	UR	C	The references listed after this message are still undefined after the last library search. The next control command is read.
LDERR	TA	C	A $X control command without transfer address has been read and no transfer address was encountered in the loading of any module. The next control command is read; it should be "!$XR transfer addr".
LDERR	MD	C	Multiple external definitions have been encountered in loading. The first encountered value will be used throughout the loading.
LDERR	SE	C	A severity level (n) greater than that specified by the $X card has been encountered in loading. A new control command is read.
LDERR	IO	A	Irrecoverable I/O error.
LDERR	EF	A	An illegal EOF has been encountered on CC, BI, or LI.
LDERR	MP	C	The $MP or $ML command did not immediately follow the LOAD command. The next command is read.

5. BCM SYSTEM LOADER

INTRODUCTION

The BCM System Loader is an extended version of the Linking Loader that permits preparation of BCM systems from relocatable decks. That is, in addition to the capabilities of the Linking Loader, the System Loader provides a method for preparing foreground tasks, background programs, an absolute FORTRAN Run-Time package, and processors for loading and execution under the BCM (resident) Absolute Loader.

FUNCTIONS

Except for the restrictions given below, the System Loader has all of the capabilities of the Linking Loader and, in addition, has the extended capabilities defined in the following list of functions:

1. Longer programs are loaded in segments as though execution memory extends beyond the background space available for loading. Essentially, the execution location counter is incremented normally, while the load location counter is incremented modulo the segment length.
2. An absolute execution location counter (foreground or background) can be specified.
3. A COMMON base can be stipulated.
4. Entries can be made to the loader symbol table by control cards.
5. Areas of memory can be punched on BO with an identifier, beginning location, end location, and transfer address specified. No severity level is output.
6. An Absolute Run-Time deck consisting of an absolute binary module and a module of external definitions (DEFs) contained in the Run-Time can be punched using the L:A option.

FORTRAN ABSOLUTE RUN-TIME

The Absolute Run-Time deck and the necessary DEF deck can be punched out by a modification to the System Loader (see SLOAD below for SLOAD command modifications via the L:A option). The L:A option indicates an Absolute Run-Time deck is to be output followed by the appropriate DEF deck.

If the L:A parameter is present, the System Loader enters a library type loading mode and all object decks following the $SL command are automatically loaded beginning at the start of background. The last object deck must be followed by an EOD command. Normally, no library programs would contain a transfer address, and the System Loader (in this case) would not consider this an error.

An $ID command will have to be used with an ident of L:A. The name L:A is a special flag to the BCM Absolute Loader that allows a transfer address of zero and will not destroy the transfer address left by the Linking Loader. If a transfer address is present in one of the Absolute Run-Time routines, it will be used and will supersede the transfer address left by the Linking Loader.

Upon encountering a $PA command, the System Loader punches out the Absolute Run-Time deck (followed by a blank card), plus all the DEFs defined in the loading process in the standard Address Definition (type 9) Object Language format. The deck of DEFs can then be loaded as a separate library program along with the user's program. The System Loader will also print out the size of the Absolute Run-Time program at the end of the run.

Figures 7 and 9 illustrate the job setups for preparation and execution of an Absolute Run-Time program. A further discussion of FORTRAN Absolute Run-Time is given in Chapter 6 of the Sigma 2/3 Basic FORTRAN Operations Manual.

RESTRICTIONS

In contrast to the Linking Loader, there are three restrictions in the use of the BCM System Loader:

1. The Linking Loader bound parameter on load commands is not honored.
2. No program execution is allowed.
3. No ASECTs will be loaded.

CONTROL COMMANDS

In general, the same rules that apply to control command format for the BCM Linking Loader are also applicable to the System Loader.

The explanation of each parameter is followed by the default case (default cases apply only for empty fields). If the field is in error or a parameter value is undefinable, a control command diagnostic will be output.

SLOAD The SLOAD command causes the BCM to execute the System Loader. It must have been previously loaded by an ABS command (see BCM Control Commands). The SLOAD command has the form

```
!SLOAD [proglim, L:A]
```

where

proglim is the total size of the area for execution. It defines the upper-limit for number of locations punched and is of the form

symbol±value
symbol
±value

where

symbol is defined as an external definition.

value is a hexadecimal number.

The default is the available background.

L:A specifies that an Absolute Run-Time package is to be punched.

$SL In the normal mode (no L:A parameter), the $SL command loads a single module into background memory. If an L:A option is given, all modules down to EOD are loaded with a single $SL command. Relocatable modules are loaded contiguously, starting at K:BACKBG+X'20'. The $SL command has the form

```
!$SL exloc, cbase
```

where

exloc is the execution location for this module.

The default is K:BACKBG+X'20'.

cbase is the COMMON base (absolute address), which must be above the user's program in core.

The cbase parameter must be specified if COMMON is used.

Here, exloc and cbase have the same form as proglim in the SLOAD command.

$LB The $LB command loads a library selectively. The $LB command has the form

```
!$LB
```

The cbase value from the $SL command is retained. The selective loading process terminates on (1) encountering an EOD, (2) satisfying all external references (the next control command is then read). One library is scanned for each $LB command. An $LB command must be preceded by an $SL command.

$ID The $ID command specifies an identification to be punched into the start item of the absolute module output. The $ID command has the form

```
!$ID idnt
```

where

idnt is the identification. It is from 3 to 8 EBCDIC characters in length.

$PA The $PA command punches an area of background memory on BO in absolute format (subset of the standard object language) that can be loaded by the resident BCM Absolute Loader. In the L:A mode, the Run-Time package (absolute module and DEFs) are punched. The identification in the start item is obtained from the $ID command. The $PA command has the form

```
!$PA begloc, endloc, transadr
```

where

begloc is the beginning of the area to be punched.

The default is K:BACKBG+X'20'.

endloc is the end of the area to be punched.

The default is the highest location loaded.

transadr is the execution transfer address to be output in the end item.

The default is the last transfer address encountered.

Note: If the value "endloc-begloc" is greater than "proglim" in the SLOAD command, only the number of cells indicated by proglim will be punched. Beloc, endloc, and transadr have the same form as proglim on the SLOAD command.

After PA has been executed, control is returned to BCM.

$DF The $DF command defines one or more symbols and enters them into the Loader symbol table. Up to 5 symbols and their values may be entered on a single card. The values override previous definitions in the symbol table. The $DF command has the form

```
!$DF symbol, value [, symbol, value...]
```

where

symbol is up to 8 alphanumeric characters in length.

value is the definition address for the symbol.

Value has the same form as proglim in the SLOAD command.

$MD, $MP, $ML These commands have the same function and format as for the BCM Linking Loader.

!EOD is used to specify the end of a Library in normal mode. Under the L:A option, !EOD defines the end of the deck to be loaded the $SL command.

ABSOLUTE RUN-TIME JOB SETUP

A typical job setup for the System Loader to punch out an Absolute Run-Time and deck of DEFs is illustrated in Figure 9 below. The Absolute Run-Time package would be reloaded subsequently by the Linking Loader (which would overlay itself as illustrated in Figure 1) to conserve memory space.

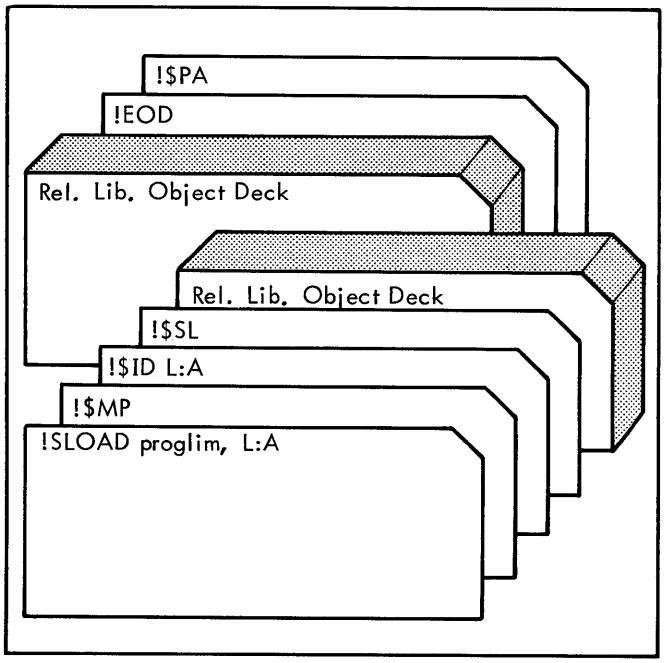


Figure 9. System Loader Job Setup for Absolute Run-Time Output

ERROR MESSAGES

Under the System Loader, the error messages in Table 3 are applicable in addition to those under the Linking Loader as given in Table 2.

Table 3. System Loader Diagnostic Messages

Message	ID	Abort/ Continue	Explanation
LDERR	CH	A	An attempt was made to resolve a forward chain outside the current segment area.
LDERR	SS	C	The area to be punched exceeds proglim. No data beyond this point will be punched.
LDERR	US	C	An undefined symbol was used on a control card. The whole card is ignored. Read the next control card. An undefined reference was found in creating an Absolute Run-Time.
LDERR	ID	A	No ID command preceded first segment punched. An ID command must be present.
LDERR	UR	A	An unsatisfied reference remains after loading an Absolute Run-Time. The Run-Time is not punched.
LDERR	IT	A	An attempt was made to load an ASECT program.

6. MONITOR SERVICE ROUTINES

BRANCHING TO SERVICE ROUTINES

Under the BCM, foreground and background programs can call the Monitor to perform various services or privileged operations. (Table 6 shows which are available.)

For requests from the background, the branch to protected memory causes an interrupt to the protection routine where the branch is examined 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 that gives the location from which the branch was attempted. If the branch is valid, the protection routine will effect the branch to the appropriate Monitor service routine.

All of the service routines are completely reentrant; therefore, they can be used by multiple tasks on a completely independent basis. Table 6 shows which of the routines require temporary space in the Monitor portion of the user's temp stack.

There are two different methods of branching to one of the Monitor service routines: one method is to branch indirectly through the address literal in the zero table, using the absolute address shown in Table 6. This is especially useful for a foreground program that is to be assembled in absolute format, a processor, or self-relocating program.

A more conventional method of branching is to declare the routine name as an external reference and let the Linking Loader satisfy the reference at load time. In this case, the address literal will be in the user's program, and it is filled in by the Linking Loader.

The B register is always saved and restored and is used as the pointer to the temporary storage.

Table 4. BCM Zero Table

Address		Name	Purpose and Assignment
Dec.	Hex.		
1	0		Reserved for Monitor Use
1	1	K:AC	Pointer to Floating Accumulator
2	2	K:AC1	Pointer to Floating Accumulator (1)
3	3	K:AC2	Pointer to Floating Accumulator (2)
4	4	K:AC3	Pointer to Floating Accumulator (3)
5	5	K:FFLG	Pointer to Floating Flags
6	6	K:BASE	Pointer to Task Reentrant Temp Stack
7	7	K:DYN	Reserved for RBM
8	8	K:LIM	Reserved for RBM
9	9		Standard Constants for Foreground, Monitor, and Background Use (see Table 5 for complete list)
.	.		
.	.		
63	3F		
64	40		IOCS Pointers and Constants
.	.		
.	.		
95	5F		
96	60		Loader and Control Card Interpreter Pointers and Constants
.	.		
.	.		
127	7F		
128	80		Real-time Foreground User Storage (Reserved for Foreground) (For communication between foreground and background or for address literals or constants)
.	.		
.	.		
.	.		
159	9F		
160	A0		Reserved for Monitor User
.	.		
.	.		
199	C6		
200	C7		Monitor Service Routines Transfer Vectors (see Table 6 for list)
.	.		
.	.		
231	E7		
232	E8		Monitor Constants (see Table 7 for a complete list)
.	.		
.	.		
.	.		
255	FF		

Table 5. Standard Constants

Address		Value		Address		Value	
Dec.	Hex.	Dec.	Hex.	Dec.	Hex.	Dec.	Hex.
9	9	32768	8000	36	24	-7	FFF9
10	A	16384	4000	37	25	-8	FFF8
11	B	8192	2000	38	26	9	9
12	C	4096	1000	39	27	-9	FFF7
13	D	2048	800	40	28	10	A
14	E	1024	400	41	29	-10	FFF6
15	F	512	200	42	2A	11	B
16	10	256	100	43	2B	-11	FFF5
17	11	128	80	44	2C	12	C
18	12	64	40	45	2D	-12	FFF4
19	13	32	20	46	2E	13	D
20	14	16	10	47	2F	-13	FFF3
21	15	8	8	48	30	14	E
22	16	4	4	49	31	-14	FFF2
23	17	2	2	50	32	15	F
24	18	1	1	51	33	-15	FFF1
25	19	0	0	52	34	-16	FFF0
26	1A	-1	FFFF	53	35	32767	7FFF
27	1B	-2	FFFE	54	36	32512	7F00
28	1C	3	3	55	37	33023	80FF
29	1D	-3	FFFD	56	38	65280	FF00
30	1E	-4	FFFC	57	39	255	00FF
31	1F	5	5	58	3A	61440	F000
32	20	-5	FFFB	59	3B	3840	0F00
33	21	6	6	60	3C	240	00F0
34	22	-6	FFFA	61	3D	49152	C000
35	23	7	7	62	3E	31	1F

Table 6. Transfer Vector for Monitor Service Routines

Address		ADRL for	Purpose of this Routine	Temp Space Required (Dec.)
Dec.	Hex.			
199	C7	M:FSAVE	M:SAVE Function if Registers Saved	0
200	C8	M:IOEX	Device-Dependent I/O Driver	16
201	C9	M:READ	Device-Independent Read Routine	30
202	CA	M:WRITE	Device-Independent Write Routine	30
203	CB	M:CTRL	Device-Independent Control Routine	30
204	CC		(Reserved for RBM use)	
205	CD	M:TERM	Normal Termination of Background	0
206	CE	M:ABORT	Abnormal Termination of Background	0
207	CF	M:SAVE	Save Registers on Real-Time Interrupt	0
208	D0	M:EXIT	Restore Registers on Foreground Exit	0
209	D1	M:HEXIN	Hexadecimal-to-Integer Conversion	0
210	D2	M:INHEX	Integer-to-Hexadecimal Conversion	0

Table 7. Monitor Constants

Address		Name	Purpose and Use
Dec.	Hex.		
232	E8	K:FOREBG	Beginning of resident foreground
233	E9	K:FOREND	Ending of resident foreground
236	EC	K:BACKBG	Beginning of background space (core)
237	ED	K:UNAVBG	Beginning of unavailable memory
239	EF	K:FEF	FORTRAN error severity level
243	F3	K:MTMP	Size of temp for Monitor service calls
244	F4	K:CCBUF	Address of control card image (buffer)

SERVICE ROUTINES

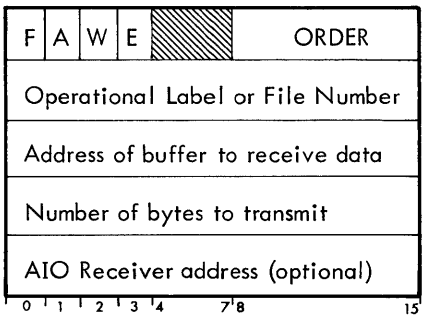
M:READ (General Read Routine)

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

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

where

ADRLST is a pointer to the first word of the argument list which is a set of four or five words either in the user's program, or in a temporary stack. This argument list appears as:



where

F = 1 A device-file number is specified.

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

A = 1 An AIO receiver address is specified.

A = 0 No AIO receiver address is specified.

(Note: Only a foreground task may specify this.)

W = 1 Unconditional wait for input to complete is specified.

W = 0 Only initiate input and return. Return is immediate if input cannot be started immediately.

E = 1 Standard error recovery is to be performed at channel end for this operation.

E = 0 No error recovery is to be attempted.

ORDER is one of the permissible "pseudo" input orders shown below:

ORDER (Hex.)	Operation
00	Return standard record size for this device in the X register and device type in A register.
02	Read binary.
04	Check previous input for completion.
06	Read automatic,
0C	Read backward.

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

The E, A, and X registers all contain status on the return, as shown in Table 8.

The return is immediate if there is a calling sequence error (and the E register is negative on return). In those cases where a wait is specified, the I/O is initiated and the M:READ routine loops until the operation is complete.

If "initiate I/O and no wait" is specified, an SIO is issued before the return if the device

1. Is currently free.
2. Can accept an SIO.
3. Is not in the "manual" mode.

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

Operation	Major Status	Action	E Reg.†	A Reg.†	X Reg.
All operations	Operational label not valid	Return immediately	-1	8	*
	Calling sequence error	Return immediately	-1	4	*
	Operational label is set to zero	Return immediately	0	2	*
	Irrecoverable I/O error	Return after error recovery attempt, if any	-1	1	*
Initiate I/O and no wait	Channel and device are free and in automatic mode	Initiate I/O and return	0	0	*
	Channel and/or device are busy	Return immediately	0	-1	*
	Manual intervention is required (manual mode or no device recognized)	Return immediately	-1	-1	*
Check and no wait	I/O still in progress	Return immediately	0	-1	*
	I/O complete	Return after end-action, if any	0	Completion code	Byte count transmitted
Initiate and wait	Channel and device are free and in automatic mode.	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 "FAULT" 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	0	Completion code	Byte count transmitted
	I/O complete	Perform any end-action and return	0 or -1		

*Unspecified.

†See Table 9 for meaning of codes shown below.

If any of these conditions is false, the M:READ routine returns immediately with appropriate indicators set.

The caller can loop back to the call to M:READ if the channel or device are busy, or can switch to another device. The "wait" flag has meaning whether this is an initiate or a check order. If error recovery is specified, it is attempted prior to the final return.

The status table (Table 8) refers to various return codes for recognition, rejection, initiation, and completion. The return codes are listed in Table 9.

On a Check operation, the byte count returned in the X register is not meaningful if the calling sequence does not specify the same count as the initial Read. This is because the byte count is calculated by subtracting the byte count residue in the status table from the byte count in the calling sequence.

M:READ FUNCTIONS

M:READ reads one physical record from the specified device regardless of device type and regardless of whether the record is EBCDIC or binary. Thus, M:READ sets 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). Fewer bytes than are in the physical record can be requested, and only the requested number will be returned in the user's buffer. However, if more bytes are requested than are in a physical record, only one physical record will be read.

In any case, the actual number of bytes read will be returned in the X register on input completion and, if this is not identical to the number of bytes requested, there will be an "unusual end" condition with an "incorrect" length code. It is not always necessary for the user to check all of the possible, distinct return codes but it can be useful to print the codes out for debugging purposes.

Table 9. I/O Completion Codes

Code in E Reg.	Value in A Reg.	Meaning	Interpretation or Effect
0	0	Operation successful	X register contains the count of the number of data bytes transmitted.
-1	1	Irrecoverable I/O error	If error recovery was specified, the maximum number of retires has been attempted unsuccessfully.
0	2	Operation not meaningful for this device	Either an operational label is assigned to file zero or the I/O operation has no meaning for the particular device.
0	3	End-of-file encountered on a Read	A tape mark has been encountered while reading from magnetic tape, or an EOD has been read from cards, paper tape, or keyboard/printer while reading in the automatic mode. (For magnetic tape, tape marks are recognized in all modes.)
0	4	End-of-tape encountered	Significant only for magnetic tapes. Tape is positioned beyond the end-of-tape marker and a write or read with the E bit = 1 has been attempted. No data is written (only tape marks can be written beyond the end-of-tape marker). On a read, if an end-of-file and an end-of-tape are both sensed at the same time, the end-of-tape return is given.
0	5	Incorrect record length specified	Significant only for read operations. For magnetic tape, the byte count and physical record size do not agree. For cards or paper tape, a read Automatic has read a binary record and the byte count is not 120, or an EBCDIC record has been read but the requested byte count was not 80.
0	6	No I/O pending for this Check operation	Probably an error in I/O buffering operations. Can be ignored, if the user desires.
0	7	Device is write protected	Significant only for magnetic tape and only for the no-wait case when a Write was attempted.
0	8	Device is at load point	Attempt to space or read backward over the load point on magnetic tape.

Using M:READ, a user can (for example) read 80 EBCDIC bytes either from cards, paper tape, or keyboard/printer. M:READ will perform standard editing from paper tape, as described below, to make the record appear as though it had come from cards.

By using a "Read and No Wait" followed later by a "Check for Input Complete" the user can effectively overlap input and computation.

The order code X'00' is used to determine the standard record size for an unknown device and can be helpful for determining the optimum blocking sizes to use.

The "Read Backward" order is required primarily as a logical backspace for Basic FORTRAN when using SDS 9-track magnetic tapes, since a logical record may be composed of several physical records. By interpreting the leading or trailing bytes on the record, FORTRAN run-time routines can determine if it is the final physical record in a logical record. (See Chapter 8 for a description of FORTRAN "unformatted" records.)

Read Backward is ignored for all devices except 9-track magnetic tape. The buffer is in an "inverted" condition after a read backward.

Note: If an "odd" byte count is given, the first byte is transmitted into the right half of the first word in the user's buffer, and the buffer will end on a word boundary (right-justified). Generally it is better to give even byte counts, because of the method used by Sigma 2/3 hardware to perform I/O operations.

REAL-TIME PRIORITY

All of the I/O routines are reenterant, and any input being initiated by a given task can be interrupted up to the "point of no return" to initiate input for a higher priority task.

External and internal interrupts are inhibited for up to 100 microseconds of CPU time during the actual SIO sequence. By keeping a high-priority task active and looping on an input request to a busy device, a high-priority task can seize control of the channel or device as soon as the current I/O operation is completed.

SPECIAL EDITING FOR READ AUTOMATIC

Card Reader. Any cards with a "1" and "2" punch in column 1 are automatically read as binary; all other cards are read as EBCDIC. (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 M:READ to EBCDIC before being returned to the user. This would apply to only one file, so it is possible to read a number of cards in EBCDIC and others in BCD from the same card reader. (BCD card codes are produced on an IBM 026 keypunch, and EBCDIC on an IBM 029 keypunch.) The EBCDIC record size is 80 bytes; the binary size is 120 bytes. An incorrect length status is returned if the requested byte count does not exactly match.

Paper Tape or Keyboard/Printer. All input from paper tape or keyboard/printer is intiated 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 from paper tape input), the M:READ routine switches to a "binary record" mode and reads up to 119 more bytes for a total of 120 in this record. The code byte is 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) and are "supersets" of the card reader "1 and 2 punch" rule for column 1. Therefore, the same codes for Read Automatic can be used for both the card reader and paper tape. 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 (NL) code is encountered. When the (NL) is encountered, input transmission is terminated and the line image is filled out (to the byte count that the user requested) with blanks. The (NL) code is not transmitted to the user buffer. (If a (NL) code is the first code in the input line, it is ignored.)

Thus, all EBCDIC records are of variable length, up to the maximum size requested (which must be no more than 80) or until a (NL) is encountered. The EOM and "cent" (¢) sign 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 thrown away. A ¢ sign acts like a backspace: for each ¢ sign received, this byte and the byte just preceding it in the buffer are discarded.

When reading binary records in the automatic mode, the paper tape is positioned after 120 bytes at the completion of the read operations, regardless of the number of bytes requested unless the record is in FORTRAN binary form. In this case, the number of bytes specified in the record are read. For either EBCDIC or binary records, no more bytes than those requested are ever transmitted to the user's buffer. For EBCDIC records, the paper tape is positioned just after the NEW LINE code. The requested byte count must be 80 for EBCDIC records and exactly 120 for binary records. Any other byte counts result in an incorrect length status being returned.

Magnetic Tape. No editing is performed for magnetic tape. Records can be of any size, since the record is always transmitted directly to the user's buffer. Binary and EBCDIC modes are identical on 9-track tape.

Only the packed-binary mode is supported by M:READ for 7-track tapes, so the mode is identical to that for 9-track tapes. However, only one physical record is read in response to one call to M:READ, regardless of the byte count.

End-of-File Marks. From the card reader, paper tape, or keyboard/printer, an EBCDIC record beginning with EOD

[†]File must be declared as BR nn at system initialization.

causes an end-of-file return from M:READ. From magnetic tape, a tape mark will cause an end-of-file return.

Device End. From magnetic tape, sensing an end-of-tape mark will cause an end-of-tape return. (Data may still be read from magnetic tape.)

Read Binary. No editing is performed, and M:READ reads the number of bytes specified (up to the physical record size of the device) as follows:

Device	Physical Record Size
Cards	120 bytes.
Paper Tape	No limit. Also, the read order is Read Immediate and the leader is read as zeros.
Magnetic Tape	Variable.
Keyboard/Printer	Variable, (up to 255 bytes)

A summary of M:READ I/O operations is given in Table 10.

Table 10. M:READ I/O Operations Summary

OP Code (Hex.)	Keyboard Printer	Paper Tape	Card Reader	Magnetic Tape
Read binary (02)	Read without edit	Read no. of bytes specified (even nos.)	Read 1 card up to 120 bytes	Read no. of bytes specified
Read automatic (06)	Read and edit to NEW LINE	Read and edit to NEW LINE	Read automatic 80 or 120 bytes (one card)	Read no. of bytes specified or one physical record, whichever is smaller
Read backward (0C)	NOP[t]	NOP[t]	NOP[t]	Read backward (one record at most)

[t]NOP indicates no operation is performed and a completion code of zero is returned in the E register and a value of 2 in the A register.

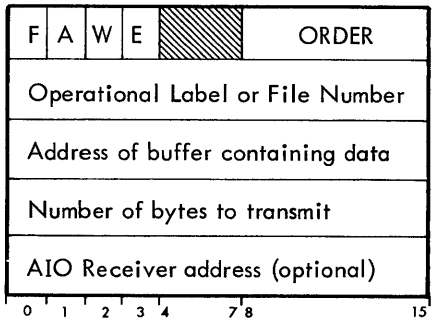
M:WRITE (General Write Routines)

The M:WRITE routine provides device-independent output with standard editing, standard error detection, and standard error recovery. The error detector is optional on each call to M:WRITE. The calling sequence is

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

where

ADRLST is a pointer to the first word of the argument list which is a set of two to five words either in the user's program, or in a temporary stack. This argument list appears as:



where

F = 1 A device-file number is specified.

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

A = 1 An AIO receiver is specified.

A = 0 No AIO receiver is specified.

W = 1 An unconditional wait for output complete.

W = 0 Only intiate output and return. Return is immediate if output cannot be started immediately.

E = 1 Standard error recovery is to be performed at channel end for this operation.

E = 0 No error recovery is to be attempted.

ORDER is one of the "pseudo" order bytes shown below.

ORDER (Hex.)	Operation
00	Return standard record size for this device in X register and device type in A register.
01	Write binary.
03	Write tape mark or EOD.
05	Write EBCDIC.
04	Check for output complete (after a "nowait" initiation).

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 is identical to that described under "M:READ".

M:WRITE FUNCTION

M:WRITE writes one physical record on the device specified, regardless of 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 other devices, the difference is not meaningful).

For devices such as a 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).

For a write binary on the keyboard/printer, the user must insert his own NEW LINE codes, as desired.

Most of the general comments applicable to M:READ also apply to M:WRITE.

Note: If an "odd" byte count is given, the first byte written is from the right half of the first word and the byte in the left half is ignored. This can be useful in writing out messages created from "TEXTC", or where the first byte is control information and, therefore, should not be written out. 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".

A summary of M:WRITE I/O operations is given in Table 11.

SPECIAL OUTPUT EDITING

Keyboard/Printer. The first byte is assumed to be a carriage control byte and is never printed. If it is an EBCDIC 0 or 1, double-spacing is used. If it is an EBCDIC minus (-), no spacing is used. Otherwise, single-spacing is used and the first byte is not sent to the keyboard/printer. Trailing blanks are removed.

If there are more than 85 printable characters, those beyond 85 are ignored.

Paper Tape. If the EBCDIC mode is specified, trailing blanks are removed. An (NL) code is always inserted as the last byte (if not already present). For the binary mode, the record is punched with no changes.

Magnetic Tape. Variable-length records are possible. No check is made of the data and no editing is performed. The exact count specified is always written.

On a Read or Write, if the tape is positioned past the end-of-tape (EOT) marker and error checking is specified, unusual end '4' is returned in A and no I/O takes place. If error checking is not specified, the Read or Write is permitted, and no EOT status returned. A Write File Mark operation is permitted regardless.

Card Punch. For the EBCDIC mode, 80 bytes are always written. The user can specify up to 80 bytes.

If this file has been declared BCD at system initialization, all EBCDIC output records are converted to BCD before being punched.

For binary write requests, 120 bytes are always punched. The user can specify from 2 to 120 bytes, and the Monitor's buffer is filled with zeros (up to 120 bytes). No modification is performed in the user's actual data or buffer.

Line Printer. The first byte per record is always assumed to be a carriage control (format) byte, and is never printed. If 133 bytes are sent, 132 will be printed and the first byte is interpreted as the format byte. With any "odd byte count"

Table 11. M:WRITE I/O Operations Summary

OP Code (Hex.)	Keyboard/Printer	Paper Tape	Card Punch	Line Printer	Magnetic Tape
Write binary (01)	Type without editing	Write even number of bytes always	Write 120 bytes always	NOP[†]	Write number of bytes specified
Write end-of-file (03)	NOP[†]	Write !EOD	Write !EOD	NOP[†]	Write tape mark
Write EBCDIC (05)	Edit and type	Edit and punch	Write out 80 bytes	Format and print up to 132 bytes	Write number of bytes specified

[†]NOP indicates no operation is performed, and a completion code of 0 is returned in the E register and a value of 2 in the A register.

(as in all I/O), the first byte transmitted is from the right 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. No incorrect length status is ever returned. If more than 133 bytes are specified, the remainder are ignored beyond 133. If fewer than 133 bytes are specified, the right-hand portion of the printed image will contain blanks.

However, the user's buffer is not modified. The print routine data chains on the order byte and format byte, first in the Monitor area, and then on the user's print image.

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

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 is treated like a blank but is not printed. These are standard FORTRAN format characters with the exception of "-". This code is substituted for the standard FORTRAN "+" to allow overprinting, but the meaning has been changed to correspond to the effect of post-slew printers. The user can use M:IOEX (General I/O Driver) and send the actual format code for any of the other format codes for SDS printers.

Write End-of-File. Order code X'03' will produce the following results on the devices given below.

Device	Result
Line Printer	No effect
Keyboard/Printer	No effect
Card Punch	!EOD card punched
Paper Tape Punch	!EOD ㊐
Magnetic Tape	Tape mark

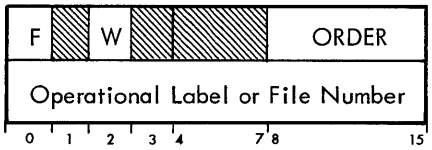
M:CTRL (General Control Routine)

This routine provides device-independent positioning capabilities for magnetic tapes (7-track and 9-track). The calling sequence is

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

where

ADRLST is the pointer to the first word of the argument list which is a set of two consecutive words in either the user's program or in a temporary stack. This argument list appears as:



where

F = 1 a device-file number is specified.

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

W = 1 wait for operation to be completed (for a rewind, complete means order was accepted and rewind has begun).

W = 0 no "wait for operation to be completed" is specified.

ORDER is one of the following "pseudo" order bytes.

ORDER (Hex.)	Operation
EB	Space Record Backward
EF	Space Record Forward
FB	Space File Backward
FF	Space File Forward
2B	Rewind (Off Line)
3B	Rewind (On Line)

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, as for M:READ, and the effect of the wait is identical to M:READ.

M:CTRL FUNCTIONS

The device (if a magnetic tape) is positioned as indicated by the order in the argument list. Error checks are not meaningful. The rewind or backspace commands are not considered an error if the tape is already at the load point.

The Space Record commands are for physical records and are not applicable for FORTRAN logical records.

The "rewind off line" operation is useful when the user wishes to save a tape off line or for a tape at end-of-reel status when a new tape must be mounted; however, the user must control and check for the end-of-reel condition. (The end-of-reel status is returned as Completion Code 4.)

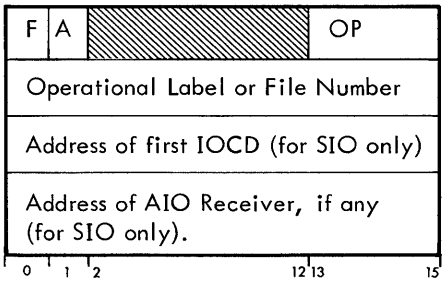
M:IOEX (General I/O Driver)

M:IOEX provides direct control by the background programs, the Monitor, or the foreground real-time programs for all I/O operations on the buffered I/O channels and additionally provides for centralization of I/O interrupts. The calling sequence is

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

where

ADRLST is a pointer to the first word in the argument list, which is a set of two, three, or four consecutive words in either the user's program or in a temporary stack. The argument list appears as



where

F = 1 A device-file number is used.

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

A = 1 If AIO Receiver is specified (foreground option only).

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

OP = Code for the operation to be performed, as:

0 for SIO

1 for TIO

2 for TDV

3 for HIO

4 for check previous data transfer

Return is to the location in the L register on the call to M:IOEX (the B register is always saved).

The Overflow and Carry Indicators, the A register, the E register, and (in some cases) the X register are used to return status information on the operation required. The complete list of status codes is shown in Table 12. In this table, the DSB stands for the Device Status Byte, the OSB for the Operational Status Byte, the Byte Count Residue is from the Odd I/O Channel Register at Channel End, and DN stands for Device Number of the current device. The "reject codes" are shown in Table 9.

Note that no I/O error recovery has been attempted. The 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 continue 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 read/write on the RAD or any peripheral device that uses standard SDS Sigma peripheral responses. For input/output operations to the RAD, the user must first give a seek order (see SDS Sigma RAD Storage System Reference Manual) 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.

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. If specified, the AIO Receiver will be ineffective.

SIO For an SIO operation, the 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. Therefore, it is the responsibility of the user's program to test for the manual mode both before and after the SIO request, and inform the operator by a suitable message.

An HIO can be used to abort an I/O operation. This results in setting the channel ready for a new activity and setting the "end action pending" flag for the device. The flag should then be cleared by an I/O check operation.

Protection checks are performed only for background I/O requests. The background is not permitted an AIO receiver, and a receiver is ignored if attempted from the background. 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 8).

Table 12. M:IOEX Return Status

Operation	Major Status	Indicators		E Register			A Register		X Register
		O	C	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 calling sequence	0	0	1			Reject code		0
SIO	SIO cannot be accepted	0	1	0	Current file number		TIO DSB	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 (DN)	Dev. no.	Byte count residue
	I/O completed normal end	0	0	0					

If the request for I/O action is for an odd number of bytes, the IOCD and order bytes must be properly set in the left or 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 data chaining, it is very important to set the interrupt flags on all IOCDs except for 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. The last IOCD must set the interrupt flag.

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. Similarly, if an AIO Receiver is specified, the branch is taken with the same status as for an ordinary return.

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 BCM I/O tables. The flag is set by the I/O

interrupt task when the device interrupt occurs. Thus, no TIOs are required in order to determine when the operation is complete.

Since TIOs take a small amount of I/O bandwidth (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.

M:TERM (Normal Exit from Background Programs)

The M:TERM routine provides a return to the Monitor on a normal termination of a background program or a foreground task-initialization program. The calling sequence is

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

M:TERM FUNCTIONS

No postmortem dumps are performed. The background is set to "empty" and all peripherals allocated to the background are now available to the foreground if there is no other activity waiting for the background job.

However, if the terminating program is one of a series of assemblies or compilations, the background is not considered "empty" and no peripherals are released or repositioned. The normal return is controlled by the Monitor and is never to the location in the L register. Nevertheless, the L register must be set as indicated.

M:ABORT (Background Abort Routine)

The M:ABORT routine provides a method of terminating the background program when it has failed for one of many possible reasons, and terminates all active I/O for the background program. The calling sequence is

```
LDA      ADRL
LDX      CODE
RCPYI    P,L
B        M:ABORT
```

where

CODE is a word of EBCDIC information (printed on the OC device) that indicates why the job was aborted.

ADRL contains the address to be printed in the error message. Return is to the location in the L register, if from a real-time foreground program. Otherwise, return is to the Monitor load routine.

Note: This routine should not be used by foreground tasks.

M:ABORT FUNCTIONS

All I/O functions in progress are allowed to complete. The actual "abort" operation is accomplished at the background priority level.

M:SAVE (Interrupt Save Routine)

This 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
         (or M:FSAVE)
ADRL     tcb
```

③

where

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

Return is to the value in the L register, plus 1. The contents of all registers are no longer in the register, but instead, are in the Task Control Block.

M:SAVE FUNCTIONS

The contents of registers A and L must be saved in the proper place in the Task Control Block before the task calls M:SAVE. M:SAVE then will save the original values of X, T, B, and E in the Task Control Block.

If the T flag of the Task Control Block is set for "temporary storage", M:SAVE will set floating accumulator pointers for the task into locations 0001 to 0005. M:SAVE will then set the task's temp stack pointer into location 0006, saving the previous contents of this location (it defines the beginning of the temp stack for the interrupted task) in the Task Control Block. If the T flag is set for "no temporary storage", M:SAVE will preserve only the hardware register for the interrupted task, and will not disturb its floating accumulator or temp stack pointers (in locations 0001 to 0006).

M:FSAVE ③

For Sigma 3 configurations a second entry point, M:FSAVE, has been added to M:SAVE for users who exercise the optional Sigma 3 Store Multiple instruction. The address literal to this entry point is saved in location X'C7' and is not available as a transfer vector to unprotected background. M:FSAVE assumes that all registers have already been saved, but it uses the same calling sequence as described above for M:SAVE.

M:EXIT (Interrupt Restore Routine)

The M:EXIT routine 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
ADRL     tcb
```

where

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

Return is to the interrupted task, at the address saved in the PSD. All registers are restored to their previous value when the interrupt occurred.

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 if the user's program can perform the exit sequence.

The Task Control Block contains a flag to indicate whether any temporary storage is used. If the task uses no Monitor I/O routines and does not use the floating accumulator, no temporary storage is needed. In this case, only the hardware registers are restored.

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 B register is unchanged. The X register is changed.

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: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 parts of a possible four-byte field, respectively).

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.

7. REAL-TIME PROGRAMMING

SCHEDULING RESIDENT FOREGROUND TASKS

The orderly transfer of control from one task to another is called scheduling. Under the BCM, with only resident foreground tasks, the scheduling proceeds in the following sequential steps.

1. When there is no background or foreground task active in the system, the Monitor enters the "wait" state and waits for the operator to direct the loading of a set of control commands from some input device.

2. When a background program is finally loaded, the Monitor transfers control to the background program by an exit sequence from the BCM Control Task. During execution of the background program (if the program is waiting for I/O to complete), there can be nothing in execution in the system. That is, the Monitor makes no attempt to multiprogram to absorb idle time. If there is a resident foreground task in core that is armed and enabled, this task may receive an interrupt from some external source.

3. As soon as the interrupt is received by the CPU, the CPU will stop at the next interruptable point and follow the interrupt entry sequence procedure (described in the Sigma 2 and Sigma 3 Computer Reference Manuals). After entry to the interrupt task, this task saves the contents of any registers it will alter and then proceeds to carry out its function. (It may use the Monitor service routine M:SAVE to do this, or it may perform the saving operations in its own code.)

4. When the real-time task is completed, it restores the contents of the saved registers and exits via the standard Sigma 2/3 exit procedure (described in the Sigma 2 and Sigma 3 Computer Reference Manuals). The real-time task may use the Monitor service routine M:EXIT to restore the previous states and exit, or it may do it in its own code.

It is important to note that this is a last-in, first-out form of scheduling. The interrupting task may be interrupted at any time during execution by a higher priority task, up to the maximum number of tasks possible in the system.

A new task saves the status and register contents of the interrupted task each time, although it does not know what the previous task was. When the new task exits , it returns control to the task it interrupted. If there is an interrupt waiting between the level of the current task (which is just completing) and the interrupted task, the interrupted task will immediately again be interrupted and the new (intermediate) task will follow the same procedure. Thus, it is never necessary for any task to know what task precedes or follows it. The task merely perserves and restores the environment according to the established rules.

The "temporary stack pointer" must be switched to the currently active stack in addition to switching the floating accumulator.[†] There are Monitor service routines provided to help make this context switch. After an interrupt, the actual entry point is to the task itself rather than to the Monitor, however.

Due to the design of the hardware priority system, the Monitor is not involved in the actual scheduling.

This procedure allows the task and programs to independently control the priority of execution 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 a 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. If, as is more likely, the Write Direct is to a task at a lower priority level, the current task will then exit in a normal manner and the next lower priority "waiting" task will continue. This next lower task may be the one that just received the Write Direct, or it may be a task in a different program; the tasks in a program can be connected to any interrupt level, and need not be adjacent in the interrupt priority hierarchy. Eventually, the task that received the Write Direct will be reached, and this task will then continue the processing at that level. Thus, with a few interrupt levels, the real-time foreground programs can have an intricate scheduling scheme without BCM action.

TASK CONTROL BLOCK FUNCTIONS

The Task Control Block (TCB) is a software convention. It is a convenient means of organizing and storing information necessary to perform task entrance and exit, defining temporary space and initial arming and enabling.

The TCB is used by the Monitor service routines M:SAVE and M:EXIT and by the control command interpreter upon encountering a C: command. The actual contents of the TCB are shown in Table 13.

The TCB can be created at assembly time as a block of code contiguous to the task it describes, with address literals pointing to the temporary stack space. By use of a DATA statement, the initial code for the interrupt level state for the task interrupt level can be set.

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".

[†]The task temp pointer and the floating accumulator can be thought of as core pseudo-registers, in that they must be saved and restored just like the actual hardware registers, but only if the task is using the Monitor I/O routines or any of the standard FORTRAN Run-Time or mathematics libraries.

Table 13. Task Control Block (TCB)

Location	Contents	Set By
TCB + 0	ADRL PSD	Assembler
1	R bit For WD | T | Dedicated Interrupt Location	Assembler
2	0001 | 0 | CODE | 0000 | GROUP	Assembler
3	ADRL TEMPBASE (temporary stack lower limit address)	Assembler
4	ADRL TEMPLIM (upper address plus one)	Assembler
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 ③)
7	Contents of X register from Interrupted Task	M:SAVE (or current task ③)
8	Contents of B register from Interrupted Task	M:SAVE (or current task ③)
9	Contents of E register from Interrupted Task	M:SAVE (or current task ③)
10	Contents of A register from Interrupted Task	Current Task (on actual entry)
11	Contents of location 0006 on current entry	M:SAVE (optional)
. . .	Note: It is optional whether or not the PSD for the interrupted task is saved in locations contiguous to TCB + 11. In any case, the saved PSD is considered to be part of the Task Control Block	
PSD + 0	Interrupted Task status	Interrupt
1	Interrupted Task status	Interrupt
2	First instruction of Task	Assembler
. . .		

In Table 13 above,

PSD is the address where the Program Status Doubleword of the interrupted task is to be stored – the same address contained in the dedicated interrupt location for the interrupting task.

R bit for WD is the hexadecimal value from (0 to F) that indicates the register bit which identifies the particular interrupt level within the GROUP (the hardware block of 16 possible interrupts).

T is the flag that indicates whether the M:SAVE and M:EXIT routines should set locations 0001 to 0006; 0 means yes, 1 means no.

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

Bit "T" in word TCB + 1 is set to show 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 temp stack pointer 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. The Monitor service routines M:SAVE and M:EXIT do not themselves use any temporary storage.

When programming the task in Basic FORTRAN, the task entrance and exit must be coded in Symbol, and the TCB is set up with the task entrance procedure. The C: control command sets the pointer to the PSD into the dedicated interrupt location.

Background programs do not require a Task Control Block.

GENERATING FOREGROUND TASKS

If a foreground program is assembled in relocatable form, it can be punched out in absolute by the System Loader. That is, the program can be created in the background even though it is to be executed in the foreground. The System Loader (like the Linking Loader) has the ability to satisfy external REFs and DEFs at load time. Thus, Symbol, Basic FORTRAN, and library programs can be combined and then punched out in absolute format as one program, for subsequent loading into the foreground.

Figure 10 illustrates a sample deck structure to punch a foreground program that calls one user-coded subroutine, plus library routines.

When the foreground program TASK is subsequently loaded (see Figure 10), it will load at +2000.

LOADING RESIDENT FOREGROUND TASKS

Foreground tasks do not have to be generated and loaded at the time that the user's BCM is created. They may be generated at any time and loaded into the foreground after an operator FG response to an !!KEY-IN.

A foreground program may be connected to its interrupt in either of two ways:

1. Using the BCM C: control command.
2. Coding an initialization routine to connect the task to its interrupt. This is accomplished by using a transfer address in the task and by requesting that the BCM transfer control to this entry point after loading, via an !name control command.

If the second method is used, the user should be aware that the initialization routine is processed at background priority level and therefore can be interrupted. Also note that as soon as the initialization routine arms and enables the task, the task itself may interrupt the initialization routine before the routine has completed its function. Therefore, the user should code this initialization routine so that it completes its initialization function before it arms and enables the task. After completing the initialization function, the routine should return to the location in the L register.

The following examples illustrate the use of both methods of connecting a foreground task to its interrupt. Example 1 assumes that the control command (CC) device is the card reader. Example 2 assumes that this device is the keyboard/printer.

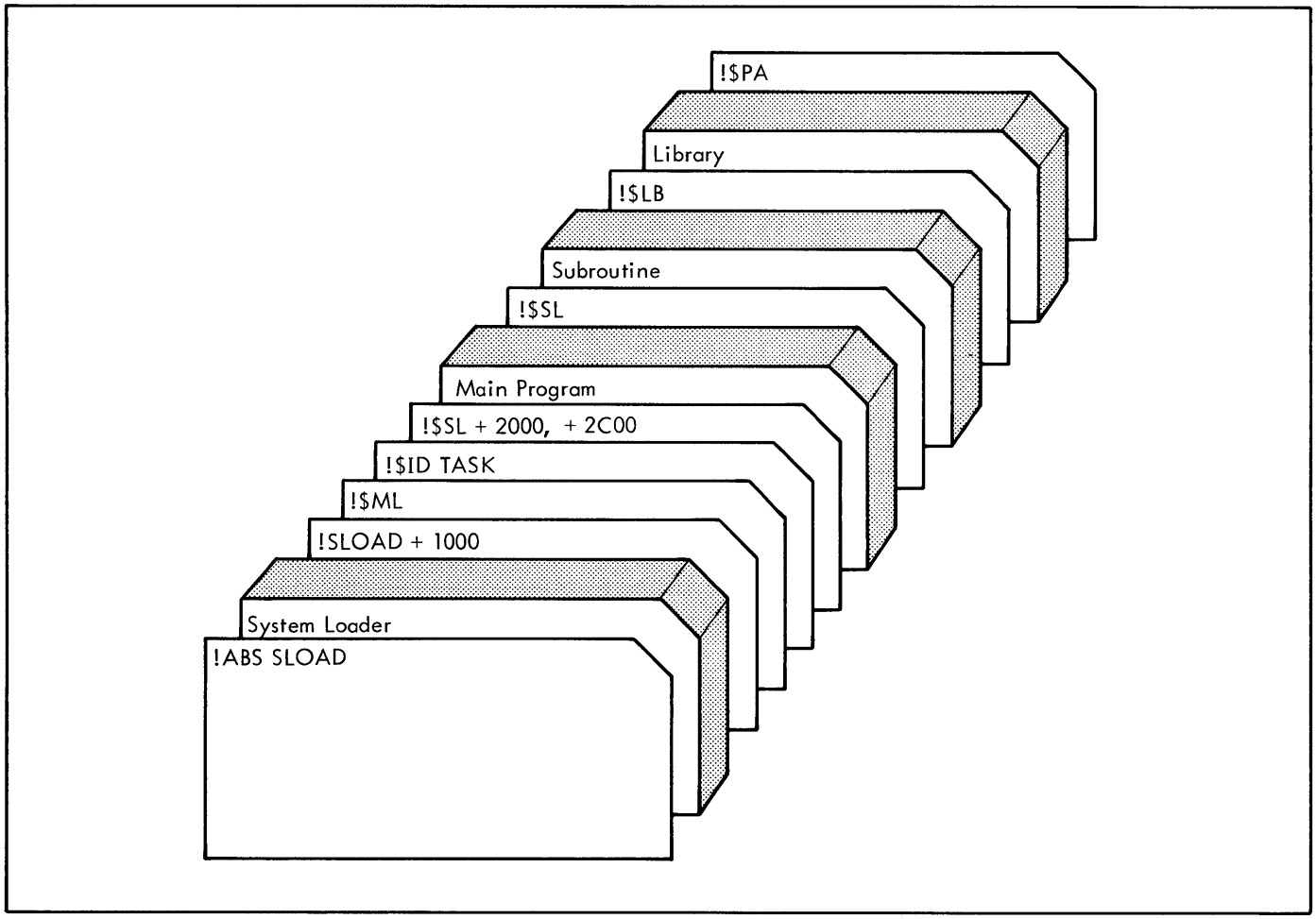


Figure 10. Deck Set-up to Generate a Foreground Program

Example 1: Using Connect Command to Connect Task to Interrupt

Activate INTERRUPT on control panel. BCM types

!!KEY-IN

Key in

FG

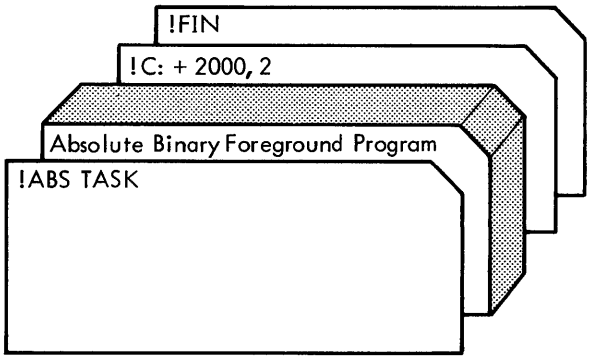
and activate INTERRUPT on control panel. BCM types

!!KEY-IN

Key in

S

BCM will now read the CC device and process the following deck:



where

TASK is the name in the start module of the foreground program; +2000 is the absolute hexadecimal address of the TCB for the foreground program.

2 is the code causing the interrupt to be armed and enabled.

Example 2: Using Initialization Routine to Connect Task to Interrupt

Activate INTERRUPT on control panel. BCM types

!!KEY-IN

Key in

FG

and activate INTERRUPT on control panel. BCM types

!!KEY-IN

Key in

S

BCM types

!!CCI

Type in

!ABS TASK

The Absolute Loader will load the foreground task and return control to the keyboard/printer. Type in

!TASK

This command causes control to transfer to the address specified in the end module of the task. On completion of initialization, the routine will return to the Monitor which will return control to the keyboard. Type in

!FIN

to deactivate the effect of the FG key-in.

FOREGROUND AND I/O PRIORITIES

All foreground programs with a priority lower than the I/O priority level may use Monitor I/O routines without difficulty or restrictions. However, real-time foreground programs with a priority level higher than the I/O priority level cannot use the Monitor I/O routines under any conditions. Thus, the termination and initiation of I/O requests must take place at a lower-than-I/O task (i.e., one that has been triggered by a Write Direct from a higher level task). Generally, the high-level tasks are used for critical real-time situations where no I/O is performed, or where the task does its own I/O due to special requirements.

8. I/O OPERATIONS

INTRODUCTION

The Monitor can perform all I/O services for the byte-oriented I/O system. This includes:

1. Logical-to-physical device equivalence.
2. Initiation of I/O requests.
3. Standard error checking and recovery (optional).
4. Software checking of background requests to preserve protection of foreground and Monitor.
5. Option to generate device order bytes to provide for device independent operations.
6. Provision for accepting user-generated IOCDs and device order bytes to provide complete control for a user's program.
7. Capability to use data chaining for foreground programs for scatter-read or gather-write operations.
8. Provision for reading or punching cards in either BCD or EBCDIC.
9. Positioning capabilities for magnetic tapes.
10. Editing capabilities from paper tape or keyboard/printer.
11. All I/O interrupt handling.

I/O INITIATION

Whenever a task needs to initiate an I/O operation, it calls on the appropriate Monitor I/O routine (see Chapter 6 for complete calling sequences). These Monitor I/O routines are reentrant, so that a higher priority task may interrupt and request I/O during a lower priority task. In this case, the lower priority task is suspended and the higher priority task is satisfied first. Thus, a real-time foreground program can take control of a multi-device controller away from background users at the completion of any current I/O operation. This technique is used in place of queuing. All Monitor I/O initiation is made according to the priority of the calling task, with background tasks having the lowest priority.

END ACTION

The section on "Operator Communication" specifies the possible error messages. Generally, error recovery takes place when I/O is checked for completion rather than on an 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 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 routine calling sequence and it is to be used only when more sophisticated end-action is required. The routine is processed at the priority level of the I/O interrupt, so the processing should be of very short duration.

Reentrancy in an end-action routine is the user's responsibility. For example, the 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 Monitor I/O task from which it originally came.

The form of the call to the AIO receiver is

```
LDA     AIODSB              (device status byte
RCPYI   P,L                 from AIO in bits 0-7,
                            device number in
B       AIO receiver address bits 8-15)
```

The AIO receiver routine should return to the location contained in the L register on entry. All registers are assumed to be volatile, which means that their contents need not be saved and restored.

The purpose of the AIO receiver technique is to allow a real-time user's program to be informed by the BCM when channel end occurs on a particular I/O operation. It is used instead of I/O queuing by the Monitor. Typically, the foreground program wishing to maximize I/O and computation overlap issues 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.

To minimize interrupt inhibit time, the channel registers are loaded and the I/O initializing SIO is issued at the I/O interrupt priority level. Consequently, any task with an interrupt priority higher than I/O cannot use M:READ, M:WRITE, or M:IOEX, but must perform its own I/O without interrupt control.

LOGICAL/PHYSICAL DEVICE EQUIVALENCE

When writing a foreground or background program in either Symbol or Basic 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 BCM to help the user in making the input/output device selection on a logical rather than physical basis. (Figure 11 gives an illustration of the process.)

The first method is called the Direct Logical Reference. The user can specify a device-file number in his calling parameters to the input/output routines, and the BCM will translate this into an actual physical device number. There may be several device-file numbers pointing to the same physical device; generally, however, only one device-file number is needed per device. For all device-file numbers, only one task may use each number. This is a necessary restriction, since the I/O status is saved in the device-file number table in the BCM, and independent operation by several tasks on the same device would cause an invalid status from the separate tasks using it. The device-file numbers are created at system initialization. The device-file number table is open-ended and any number of entries may be created.

The second method of device referencing is through the Indirect Logical Reference. The Indirect Logical Reference will first equate a device unit number or an operational label to a device-file number, which in turn is equated to a physical device number. The equivalence between operational labels or device unit numbers and the device-file numbers is set at system initialization time for certain standard devices as illustrated in Tables 14 and 15. These standard assignments can be changed later by use of assignment control commands.

The standard background operational labels are merely names. The devices and functions indicate how the standard processors use the labels. Since each I/O call must specify a byte count, a user's program can read any number of bytes from SI (if SI is magnetic tape, for example). There is no restriction on the record size except as imposed on the peripheral devices. The standard operational labels are given in Table 15.

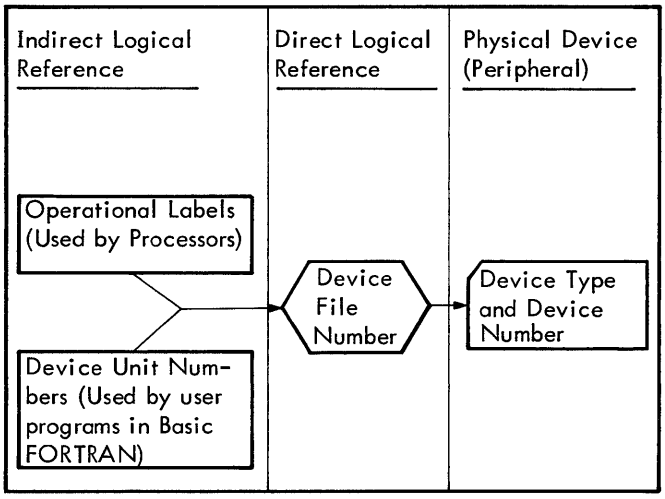


Figure 11. Logical/Physical Equivalence

Table 14. Standard FORTRAN 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 15. Standard Background Operational Labels

Operational Label	Reference	I/O Device	I/O Function
SI	Symbolic input	KP, CR, PT, MT, TY	Read number of bytes specified, up to 80.
BI	Binary input	CR, PT, MT	Read number of bytes specified, up to 120.
LI	Library input	Same as BI	
BO	Binary output	CP, PT, MT	Punch number of bytes specified, up to 120.
LO	Listing output	LP, KP, MT, TY	Perform device format operations, if required, and print up to 132 bytes.
DO	Diagnostic output	same as LO	
LL	Listing log	same as LO	
PB	Punch BCM	same as BO	
CC	Control command input	KP, CR, PT, MT, TY	Read number of bytes specified, up to 72.
OC	Operator's console	KP, TY	Write number of bytes specified.
X1	Intermediate scratch	MT	Read or write number of bytes specified.
UI	Update input	MT	Read/write as specified.
UO	Update output	MT	Read/write as specified.
AI	Absolute input	same as BI	

FORTRAN BINARY I/O RECORD FORMAT

Sigma 2/3 Basic FORTRAN binary record formats are compatible with Sigma 5/7 FORTRAN IV-H binary record formats, as follows:

Physical Record Byte	Contents
1	X'3C' – physical binary record code for any record except the last physical record of a logical record
1	X'1C' – code for last physical record in a logical binary record
2	Checksum – sum of all the bytes in the physical record excluding the checksum byte and ignoring byte overflow
3 and 4	Physical record size – number of bytes in the physical record, including the control bytes
5...n	Zero or more data bytes in the physical record
n+1	X'3C' – not the last physical record of the logical record
n+1	X'1C' – the last physical record in a logical record
n+2	X'BD'
n+3 and n+4	Physical record number – starting with zero and increasing by one for each succeeding physical record in the logical record

A logical record may be composed of one or more physical records but no more than one logical record may occupy any given physical record. If a logical record extends over more than one physical record, the control information (above) can be used to correctly read/write position the logical record.

The maximum size of a physical record depends on the device involved. (For cards or paper tape, the maximum size is a total of 120 bytes; for magnetic tape, the maximum size is a total of 360 bytes.) The "read backward" order in M:READ can be used to backspace 9-track magnetic tape over logical records without the backspace-read-backspace sequence being necessary (as with 7-track tapes). The minimum record size is 44 bytes for a read backward order.

9. UTILITY SUBSYSTEM

INTRODUCTION

The Utility Subsystem is a processor that operates in the background under the BCM. It provides the BCM user with a media copy routine, a record editor, an object module editor, a dump routine, and a sequence number editor. The routines are device independent.

CALLING THE UTILITY SUBSYSTEM

The Utility Subsystem is requested via a UTILITY control command, but it must be loaded intor core by an ABS control command prior to the UTILITY control command input. The UTILITY control command is read from the CC device and has the following format.

```
!UTILITY  [name] [,parameters]
```

where

name is the name of a Utility Subroutine or may be omitted. It may be any of the following:

COPY (copy routine)
RELEDIT (record editor)
OMEDIT (object module editor)
DUMP (dump)
SEQEDIT (sequence editor)
omitted (control functions only)

Parameters are optional and unique for each Utility Subroutine; their use is explained in the description of the individual routines.

UTILITY SUBSYSTEM RESPONSE

When the Monitor encounters a UTILITY control command on the control command (CC) device, it checks to determine if the requested processor is currently in core storage.

If the Utility Subsystem is not in core storage, the BCM follows its standard procedure when a requested processor is not loaded. If the Utility Subsystem is present, control is transferred to the Utility Subsystem Control Routine, with the X register containing the address of the control command image.

If the "name" parameter is present, the Utility Subsystem is able to identify the subroutine to be executed. Control is then transferred to the requested routine to process the parameters, if they exist. If the Utility Subsystem cannot identify the subroutine named on the UTILITY control command, the message

UT NT RES

is written on OC and DO and the Utility Subsystem aborts.

UTILITY SUBSYSTEM CONTROL

The Utility Subsystem consists of two major sections: the Utility Subsystem Control (always resident when the Utility Subsystem is operating); and the currently operating Utility Subroutine. The Utility Subsystem Control contains four interdependent elements:

- The Utility Subsystem Executive that initializes the subsystem upon entry from the BCM, interprets the UTILITY control command, exercises control over the flow of control commands, handles the normal and abort exits to the Monitor, and performs all I/O checking for the Utility Subsystem.
- The Source Input Interpreter that reads and scans Utility Subsystem control commands for the Control Function Processor and the current Utility Subroutine.
- The Control Function Processor that executes control function commands common to all Utility Subroutines (such as *REWIND and *FSKIP).
- The Operator Communication Routine that outputs messages to OC and DO, and receives key-in responses.

The various portions of the Utility Subsystem Control are described in detail below.

UTILITY SUBSYSTEM EXECUTIVE

When the Monitor has read a UTILITY control command (on CC) and has determined that the Utility Subsystem is resident, control is transferred to the Utility Subsystem Executive Routine. The UTILITY control command is then scanned for parameters. If the "name" parameter is omitted, it is assumed that the control commands on SI call only upon the facilities of the Control Function Processor.

If a specific Utility Subroutine is named, the Executive checks to see if that subroutine is in core storage: if not, an error message is written and exit to the Monitor is taken, which terminates the background operation. If the subroutine is present, initialization of tables and flags occurs.

The 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.

When the Executive or Utility Routine encounters an !EOD card image from SI, it terminates processing. The form of the EOD command is

```
!EOD
```

This causes the Utility Subsystem to transfer control back to the Monitor.

SOURCE INPUT INTERPRETER

All control commands read by the Utility Subsystem are input from the Symbolic Input (SI) device and are first processed by the Source Input Interpreter which, in turn, is called by the Utility Subsystem Executive Routine. Control commands are listed on the DO device as they are interpreted. The control commands have the general form

```
!*mnemonic specification
```

where

- ! identifies the record as the beginning of a control command.
- * indicates that the control command is unique to the Utility Subsystem.
- mnemonic is the code name of a Utility Control Command. It must begin immediately following the !* characters.
- specification is a series of required or optional parameters unique to the specific command. The following special conventions are used in specifying parameters:
  1. A string of up to 5 decimal digits, having a value less than 32,768 denotes a decimal integer.
  2. A string containing more than 5 characters is always assumed to be EBCDIC, regardless of content.

One or more blanks separate the mnemonic and specification fields, but no blanks may be embedded within a field. A control command is terminated by the first blank after the specification field; or, if the specification field is absent and a comment follows the mnemonic field, the command is terminated by a period. No control command record may contain more than 80 characters.

In the pictorial representation of control commands in this chapter, certain conventions are used to describe the formats of specification fields. Optional parameters are shown enclosed in brackets (no brackets appear in the actual control command). The omission of a parameter embedded within the field is denoted by an additional comma. A comma may not terminate a field.

The first two characters of the mnemonic portion of the command are sufficient to define a control command; the remaining characters can be omitted, since they are ignored when present. Therefore, the user may input commands in an abbreviated form (e.g., !*MODIFY may be input as !*MO). This is particularly useful when control commands are input from the keyboard/printer.

Upon reading a command, the Control Command Interpreter determines if the command is valid. If the syntax for a command is invalid, the following message is written on OC and DO:

```
INV CTRL
!!UKEYIN
```

The operator response (either an S for continue or X for abort) determines whether or not the Utility subsystem 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, which executes commands common to all Utility Subroutines (e.g., *MESSAGE, *PAUSE, etc.).

Utility functions are generally executed dynamically; that is, control commands in the SI stack are interpreted and executed as they are read. However, when the same device is assigned to several operational labels, it is impractical to execute dynamically. In this case, certain input is prestored. This decision to prestore is made by the Utility Subsystem 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.

CONTROL FUNCTION PROCESSOR

The Control Function Processor interprets and executes the commands, given below, that are common to all Utility Subroutines. If any of the control commands interpreted and executed by the Control Function Processor contain an invalid operational label (not allowed by requested Utility Subroutine) the message

```
INV OPLB
!!UKEYIN
```

is output.

In all cases, the response for invalid operational labels is one of the following, which is keyed in on OC by the operator:

- S (continue; this causes the next SI card image to be read)
- X (abort; this causes a return to Monitor control)

!*FSKIP (File Skip Forward)

The *FSKIP control command causes a magnetic tape to be spaced forward until the specified number of file marks has been passed. The form of the command is

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

where

- oplb is the operational label of the device.
- number is the number of file marks to skip. If omitted, the number is assumed to be 1.

If the oplb parameter is missing, or if the number parameter is non-numeric or greater than 32,767, the following message is written on OC and DO.

```
PARAM ERR
!!UKEYIN
```

If oplb identifies an invalid device, the following message is written on OC and DO.

```
INV OPLB
!!UKEYIN
```

If two consecutive file marks are encountered before the required number of files is passed, the following message is typed on the OC and DO devices.

```
DEOF oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If the end-of-tape is encountered before the required number of files has been passed, the following message is typed on OC and DO.

```
EOT oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

!*RSKIP (Record Skip Forward)

The *RSKIP control command causes the device to be spaced forward until the specified number of records has been passed. The form of the command is

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

where

oplb is the operational label of the device.

number is the number of records to be skipped. If it is omitted, the number is assumed to be 1.

If the oplb parameter is missing, or the number parameter is non-numeric or greater than 32,767, the following message is written on OC and DO.

```
PARAM ERR
!!UKEYIN
```

If oplb identifies an invalid device, the following message is written on OC and DO.

```
INV OPLB
!!UKEYIN
```

If an !EOD or file mark is encountered before the required number of records is passed, the following message is typed on OC and DO.

```
EOF oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If the end-of-tape is encountered before the specified number of records has been skipped, the following message is typed on OC and DO.

```
EOT oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

!*FBACK (File Skip Backward – magnetic tape only)

The *FBACK control command causes the magnetic tape to be spaced backward until the specified number of file marks have been passed. The form of the command is

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

where

oplb is the operational label of a magnetic tape.

number is the number of file marks to skip. If it is omitted, the number is assumed to be 1.

If the operational label parameter is missing or contains more than 2 characters, or if the number parameter is non-numeric or greater than 32,767, the following message is written on OC and DO.

```
PARAM ERR
!!UKEYIN
```

If the operational label identifies an invalid device, the following message is written on OC and DO.

```
INV OPLB  oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If beginning of tape is encountered before the required number of files have been passed, the following message is written on OC and DO.

```
BOT   oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If double end-of-file marks are encountered while executing *FBACK, the following message is written on OC and DO.

```
DEOF   oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

!*RBACK (Record Skip Backward – magnetic tape only)

The *RBACK control command causes the magnetic tape specified to be spaced backward the specified number of records. The form of the command is

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

where

oplb is the operational label of the magnetic tape.

number is the number of records to be passed. If it is omitted, the number is assumed to be one.

If the operational label parameter is missing or contains more than 2 characters, or if the number parameter is non-numeric or greater than 32,767, the following message is written on OC and DO.

```
PARAM  ERR
!!UKEYIN
```

If the operational label identifies an invalid device, the following message is written on OC and DO.

```
INV  OPLB   oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If a file mark is encountered before the specified number of records have been passed, the following message is written on OC and DO.

```
EOF  oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device numbers.

If beginning of tape is encountered before the requested number of records have been passed, the following message is written on OC and DO.

```
BOT   oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

!*REWIND (Rewind – magnetic tape only)

The *REWIND control command causes the specified magnetic tape to be rewound. The form of the command is

```
!*REWIND  oplb
```

where

oplb is the operational label of the magnetic tape to be rewound.

If the operational label parameter contains more than 2 characters, the following message is written on OC and DO.

```
PARAM  ERR
!!UKEYIN
```

!*UNLOAD (Unload – magnetic tape only)

The *UNLOAD control command causes the specified tapes to be unloaded. The form of the command is

```
!*UNLOAD  oplb
```

where

oplb is the operational label of the magnetic tape.

If the operational label parameter is missing or contains more than 2 characters, the following message is written on OC and DO.

```
PARAM  ERR
!!UKEYIN
```

If the operational label identifies an invalid device, the following message is written on OC and DO.

```
INV OPLB   oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

!*MESSAGE (Message)

The *MESSAGE control command writes a message to the operator on the OC and 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 (Message With Pause)

The *PAUSE control 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
```

!*WEOF (Write File Mark or EOD)

The *WEOF control command causes a single file mark to be written on magnetic tape or an !EOD source image (80 bytes) to be written on a non-magnetic tape medium. The form of the command is

```
!*WEOF oplb
```

where

oplb is the operational label of the device.

If the oplb parameter is missing, the following message is written on OC and DO.

```
PARAM ERR
!!UKEYIN
```

If oplb identifies an invalid device, the following message is written on OC and DO.

```
INV OPLB
!!UKEYIN
```

If the end-of-tape is encountered while an *WEOF command is being executed, the following message is written on OC and DO.

```
EOT oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

!*PRESTORE (Prestore SI)

The *PRESTORE control command causes all control commands and data to be read from the SI device until an !EOD is encountered. Interpretation of the control commands then begins. (NOTE: the prestore mode is set automatically where 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 precede any other control commands for the Utility Subsystem. The form of the command is

```
!*PRESTORE
```

If an overflow of available memory occurs, the following error message is typed on OC and DO.

```
CORE OVFLO
```

The Utility Subsystem aborts operation and transfers control to the Monitor.

If the *PRESTORE command is not the first command following UTILITY, the following message is written on OC and DO.

```
PRE ERR
!!UKEYIN
```

OPERATOR COMMUNICATION ROUTINE

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

If a response is required from the operator, the Operator Communication Routine types

```
!!UKEYIN
```

The operator keys in one of the following responses on OC.

S (continue processing)

X (abort the Utility Subsystem operation and return control to the Monitor)

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.

I/O ERROR MESSAGES

The Executive performs all I/O checking for the Utility Subsystem. The messages regarding I/O errors are written on both the OC and DO devices.

If manual intervention is required (the device is in manual mode or no device is recognized) the following message is written.

```
EMPTY oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

Unless otherwise noted, the operator response is

S (continue; the operator has readied the device)

X (abort)

If the operational label is not valid, the following message is written.

```
INV OPLB oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

The "oplb,device" portion of the message may contain invalid data if I/O is attempted for an operational label not recognized by the Monitor.

If an unrecoverable I/O error occurs after the maximum number of retries has been unsuccessfully attempted, the following message is written.

```
UNRECOV I/O
```

The Utility Subsystem aborts.

If an unexpected tape mark has been encountered while reading from magnetic tape or an unexpected EOD has been read from cards, paper tape or keyboard/printer, the following message is written.

```
EOF oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If the end-of-tape mark is sensed on magnetic tape, the following message is written.

```
EOT oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If an attempt is made to write on a write-protected tape, the following message is written.

```
WRITE PRO oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If an attempt is made to space backward over the load point on magnetic tape, the following message is written.

```
BOT oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If an I/O operation is not meaningful for the device requested, the following message is written.

```
INV I/O OP oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If the I/O calling sequence is in error, incorrect length is specified, or no I/O is pending for a check operation, the following message is written.

```
I/O ERR   oplb,device
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

The Utility Subsystem aborts.

ABORT RETURN TO MONITOR

When an irrecoverable error occurs, the Utility Subsystem aborts by calling the Background Abort Routine (M:ABORT). For an irrecoverable I/O error, the code in the abort message is the operational label for the device. The code is 'UT' if the abort was caused by an X response by the operator, or by some other error condition.

CONTROL ROUTINE OPERATIONAL LABELS

Four operational labels are reserved for the Utility Subsystem Control Routine and 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.

- CC Device for Monitor control command input (UTILITY and EOD control commands only).
- OC Device for messages to the operator, or key-in responses from the operator (always via the keyboard/printer).
- SI Device for Utility Subsystem control commands and various modification source inputs.
- DO Device for listing of control commands as they are interpreted, messages, error conditions, operator responses, etc. Provides a permanent log of the control command flow. This is the only operational label for Utility Subsystem control that can be assigned to the 0 (zero) device-file number (i.e., suppressed). If OC and DO are assigned to the same device, duplication of messages is suppressed.

COPY

Under the BCM, COPY provides the ability to copy variable length binary or EBCDIC records from cards, paper tape, magnetic tape, or keyboard/printer to cards, paper tape, magnetic tape, line printer, or keyboard/printer. Using control functions of the Control Function Processor, records and files can be skipped. Output generated by the COPY routine can be verified.

Since COPY uses M:READ and M:WRITE for all reading and writing, data copied with the COPY routine must be in a standard format.[†] The distinction between binary and EBCDIC modes is artificial except for the card punch.

For records being copied to the card punch, the COPY routine performs as follows. 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 the EBCDIC mode.

Note: When writing on an EBCDIC device such as the keyboard/printer or line printer, no special reformatting is performed; therefore, attempting to copy binary data to an EBCDIC device may 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', and X'78') is always 120 bytes. The BIN control card allows the user to override the standard count. 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 user can override the standard count with a control card option.

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 (i.e., where the first byte does not contain X'1C', X'3C', X'9F', X'BF', X'DF', X'78', X'00', or X'FF'). 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.

OPERATIONAL LABELS USED

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

- X1 (verify device)
- UI (input device)

[†]Certain "standard" conventions can be overridden by use of SIZE, MODE, and BIN parameters. The user should be familiar with standard conventions to ascertain the effect of the deviations. (Bootstrap records on paper tape, for example, contain 128 bytes.)

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

OPERATING CHARACTERISTICS

The COPY routine checks whether or not SI, X1, UI, and any other 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.

If the operational labels are not assigned to the same physical devices, interpretation of control commands takes place as they are read from SI.

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 devices, 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 SI and the input or output operational labels are not assigned to the same physical device, 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.

***OPLBS** (Specify Operational Labels)

The *OPLBS control command specifies the operational labels of devices to be used in *COPY and *VERIFY requests and must precede the first *COPY or *VERIFY control command (i.e., an *OPLBS command must follow the UTILITY command). These operational label assignments remain in effect until a new *OPLBS control command is read. The form of the command is

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

where

oplb$_i$ is the operational label for an output device for subsequent *COPY commands, or an input device for subsequent *VERIFY control commands. 'oplb' cannot be assigned to device-file number 0.

***COPY** (Copy)

The *COPY control command causes records from the input device (UI) to be copied on the output device(s) (specified on the *OPLBS command) until the requested number of EOD 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. If "type" is R, "number" is the number of records to be copied. 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 "number" 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 utilized, 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 "size" is omitted, all records are read and written in the standard record size (120 bytes).

BIN if omitted, allows mode (BIN or EBCDIC) to be determined according to byte 1 of the record. If BIN is present, all copying is done in binary, either with the count specified in "size" or by the standard record size (120 bytes) by default.

***VERIFY** (Verify)

The *VERIFY control command is used to request the comparison of data on the X1 device with data in core (CORE option)

or with data from devices specified on 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.

Before the *VERIFY control command is issued, it is assumed that all magnetic tape files have been repositioned, if necessary, by use of *REWIND and other file positioning control commands (described in Control Function Processor).

Any errors found by the verification process cause the message

```
VERIFY ERR oplb,device
```

to be written on OC and DO

where

oplb is the operational label of the device on which the error was detected.

device is the device type and physical device number.

When a verification error occurs, the COPY routine terminates execution of the *VERIFY command for that device, but continues verification on the other input devices.

The entire verification process is completed when the number of files or records requested for verification has been compared. If an error is detected on every input device, the Utility Subsystem is aborted. The standard BCM abort message is written on OC with a code of VE.

If an end-of-tape, two consecutive tape marks, or EODs are detected on X1 or UI before the number of files requested has been compared, the following message is typed

```
EOT oplb,device
!!UKEYIN
```

or

```
DEOF oplb,device
```

where

oplb is the operational label of a device (either X1 or UI).

device is the device type and physical device number.

The response for EOT is:

S (continue; this causes processing to continue)

X (abort)

If an EOD or file mark is detected on X1 or UI before the number of records requested have been compared, the following message is written on the OC and DO device.

```
EOF oplb,device
!!UKEYIN
```

where

oplb is the operational label of a device (either X1 or UI).

device is the device type and physical device number.

RECORD EDITOR

The Record Editor routine generates or updates tapes (paper or magnetic) containing symbolic source data. The following capabilities are provided:

1. Generates a tape containing source data.
2. Lists a tape containing source images in addition to associated line numbers.
3. Modifies tapes containing source images.

OPERATIONAL LABELS USED

The following operational labels must be assigned in addition to the standard Utility Subsystem operational labels:

SI Input device for control commands
LO Input device for listing source images
UI Input tape device
UO Output tape device

OPERATING CHARACTERISTICS

The Record Editor 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 a tape on the UO device.

The Record Editor uses M:READ and M:WRITE to perform all I/O. 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 control command

```
!UTILITY RECEDIT
```

After interpretation of the UTILITY control commands, control is transferred to the Record Editor, which begins reading control commands.

CONTROL COMMANDS

A command requesting the list or modify mode must immediately follow the UTILITY command. All other control commands are interpreted as sub-commands under each mode. If a binary record is read from UI, the message

```
MODE ERR UI,device
!!UKEYIN
```

is written on OC and DO

where

device is the device type and physical device number.

*LIST (List Mode)

The *LIST control command 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 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.

Upon entering the list mode, the Record Editor checks whether or not both SI and UI are the same device. If so, the following message is written on OC and DO.

```
!!LD LIST UI,device
```

where

device is the device type and physical device number.

The operator responds by mounting the tape to be listed and changes the state of the device. If both SI and UI are not assigned to the same device, listing begins immediately. For subsequent *LIST control commands, no message is written. A *MODIFY control command or an EOD control command causes the list mode to terminate.

*MODIFY (Modify Mode)

The *MODIFY control command informs the Record Editor that a tape is to be either generated or updated. 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 a new tape is to be generated (i.e., there is no input tape on UI) and written on the UO device. If updating is to be performed (i.e., there is an input tape on UI to read), the field is left empty.

When the modify mode is entered and updating is to be performed, the following message is written on the OC and DO device.

```
LD INPUT UI,device
!!UKEYIN
```

where

device is the device type and physical device number.

The operator must respond by mounting the tape to be input and key-in an S response on OC to continue.

The modify mode is terminated whenever a *LIST, *MODIFY, 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 are written on UO, followed by an EOD or file mark.

The modify mode control commands are *DELETE, *INSERT, and *CHANGE. If the Record Editor is not in the modify mode and one of the commands is interpreted from SI, the following message is written on OC and DO.

```
INV CTRL
!!UKEYIN
```

An automatic copy to the appropriate place on the output tape is performed preceding the execution of a *DELETE, *INSERT, or *CHANGE control command. The Record Editor remains in the modify mode until a *LIST or EOD control command is interpreted.

***DELETE** (Delete Records – modify mode only)

The *DELETE control command causes the indicated source image(s) 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. If $number_2$ is omitted, only one image is deleted.

***INSERT** (Insert Records – modify mode only)

The *INSERT control command causes source card(s) to be added to the output tape 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 should 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 Paper Tape Editor control command is encountered.

***CHANGE** (Replace Records – modify mode only)

The *CHANGE control command causes the indicated source image(s) to be deleted, and source card(s) following the *CHANGE command to be written on UO. The command is effective only in the modify mode. The form of the command is

```
!*CHANGE  number1,number2
```

where

$number_1$ is the line number of the first (or only) source image to be deleted.

$number_2$ is the line number of the last source image to be deleted. If $number_2$ is omitted, only one image is deleted.

Following the *CHANGE control command, every source image on SI is inserted until another Record Editor control command is encountered.

OBJECT MODULE EDITOR

The Object Module Editor is designed to maintain tapes containing libraries of standard Sigma 2/3 object language modules. It generates or updates tapes by inserting and deleting object modules according to the program name in the start module item for each module. For each output tape written, a list of module names is printed in the order of their appearance.

The Object Module Editor is also used to list a tape 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 library consists of one or more object modules and is terminated by a file mark or EOD. A library tape may contain one or more libraries and is terminated by double file marks or EODs.

OPERATIONAL LABELS

The Object Module Editor uses the following operational labels.

- LO Device for listing either UI or UO object module names
- BI Device from which binary object modules are to be inserted
- UI Input tape device
- UO Output tape device

OPERATING CHARACTERISTICS

If any two of the operational labels SI, BI, and UI are assigned to the same device, every control command is read in from SI and stored in memory until an EOD or file mark is encountered. The control commands are interpreted in order and written on DO.

If no two of 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.

The Object Module Editor operates in two modes: list and modify.

In the list mode, the tape on UI is read. The names of the object modules on the tape are printed on LO, and the checksum and sequence for each record are verified. After interpreting the *LIST control command, the editor checks to see 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 mounting the tape to be listed on UI and keys in an S response. Listing of the tape proceeds. If no two of SI, BI, and UI are assigned to the same device, no message is written and listing begins immediately.

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 is an input tape to be updated, the tape is 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 "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 SI, BI, and UI operational labels are assigned to the same device, the 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. Then, the Editor 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 is an input tape to be updated, the message

```
LD INPUT
!!UKEYIN
```

   is written on OC and DO. The operator must load the tape to be updated on UI and key in an S response.

3. The modify mode 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 modify mode control command. Every control command read from SI is listed on DO.

The Object Module Editor uses M:READ and M:WRITE to perform all I/O. Each object module is identified by the program name stored in the start module item. No modules with blank names are ever written on the UO tape. If any blank program names are input, the following error message is written on OC and DO.

```
BLNK NAME oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

Unless otherwise noted, the operator responses are:

S (continue; this causes the next SI card image to be read)

X (abort)

If a checksum error is detected on any of the records read from UI or BI, the following message is written on OC and DO.

```
CKSM ERR oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

The operator responses are

S (the Editor continues reading the UI or BI input and the record in error is written on UO if an output tape is being generated)

X (abort)

If a sequence error is detected on any of the records read from UI or BI, the following message is written on OC and DO.

```
SEQ ERR oplb,device
!!UKEYIN
```

where

oplb is the operating label of the device.

device is the device type and physical device number.

The operator responses are

S (the Editor continues reading the UI or BI input and the record in error is written on UO if an output tape is being generated)

X (abort)

If the first byte of a record read from UI on BI does not contain X'FF' or X'9F', the following message is written on OC and DO.

```
ILLEG BIN oplb,device
!!KEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

If two consecutive EODs or tape marks on UI or one EOD or tape mark on BI are encountered during the editing process before the desired number of modules have been copied, the following message is written on OC and DO.

```
NO name oplb device
!!UKEYIN
```

where

name is the program name not found.

oplb is the operational label of the input device.

device is the device type and physical device number.

If an end-of-tape is encountered before a single EOF on BI, or before a double EOF on UI, the following message is output on OC and DO.

```
EOT oplb,device
!!UKEYIN
```

where

oplb is the operational label of the device.

device is the device type and physical device number.

CALLING OBJECT MODULE EDITOR

The Object Module Editor is given control via the command

```
!UTILITY OMEDIT
```

The Object Module Editor begins reading control commands until an !EOD is read, which terminates the SI input.

CONTROL COMMANDS

***LIST** (List)

The *LIST control 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 EOD, it is listed. If two consecutive EODs occur, the Editor leaves the list mode and the next control command is interpreted. The form of the command is

```
!*LIST
```

***MODIFY** (Modify)

The *MODIFY control command indicates to the Editor that a library tape is 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. 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 a new tape is 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 can be used in the GEN mode.

INSERT must be specified if insertions from BI are to be read. If BI and UI are on the same physical device, the complete BI file (up to 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) the UI tape and to write a UO tape.

Note: If INSERT and GEN are omitted from the *MODIFY control command, only *DELETE control commands may be input.

***INSERT** (Insert)

The *INSERT control 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 8 EBCDIC characters) of the object module to be inserted.

$name_2$ is the name (up to 8 EBCDIC characters) of the object module on the UI tape that the $name_1$ object module must follow. If $name_2$ is omitted, the $name_1$ is written following the module previously written on UO.

Modules to be inserted from BI must be in the same order as in the *INSERT control commands. If GEN is specified on the *MODIFY command, only the $name_1$ parameter on the

*INSERT command is required; if $name_2$ is specified, it is ignored.

***DELETE** (Delete)

The *DELETE control command causes object modules to be deleted and is effective only in the modify mode. The form of the command is

```
!*DELETE  name1[,name2]
```

where

$name_1$ is the program name (up to 8 EBCDIC characters) of the first or only module on the UI tape to be deleted.

$name_2$ is the program name (up to 8 EBCDIC characters) of the last module on the UI tape to be deleted. If absent, only one module is deleted.

The *DELETE control command must name modules in the same order as the programs occur on UI.

DUMP

The Dump routine (DUMP) provides the capability of dumping tapes onto an output device in either hexadecimal or EBCDIC format.

The Dump routine uses M:READ and M:WRITE for all I/O. If no mode or the EBCDIC mode is specified for dumping, all records are dumped according to the content 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 it 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 records 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 (i.e., where the first byte does not contain X'1C', X'3C', X'9F', X'BF', X'DF', X'78', X'00', or X'FF'). The BIN 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 4-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 not recognized as a file mark. Therefore a request to dump one or more files can be terminated when the specified number of records has been dumped or by putting the device in manual mode. A request to dump one or more files (when the device is magnetic tape) is processed normally, since file marks are recognized.

OPERATIONAL LABELS USED

The Dump routine uses the following operational labels:

SI Device for input commands
UI Input device for dumping
LO Output device for dumping (unless some other input device is specified)

OPERATING CHARACTERISTICS

If both SI and the 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
!!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 PTDUMP control commands are then listed on DO and dumping is performed.

CALL DUMP

Control is transferred to the Utility Package via the control command

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

where

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

After interpretation of the UTILITY control command, control is transferred to the Dump routine. The control command and options available to DUMP are described below.

CONTROL COMMANDS

***DUMP** (Dump)

The *DUMP control command causes records to be read from the input tape and written on the LO device in the specified mode until an EOD or file mark is read. The form of the command is

```
!*DUMP  [number] [,mode] [,size]
```

where

number is a decimal integer. Only the specified number of records are dumped. If "number" is omitted, the file is dumped through an EOF or file mark. If "number" is ALL, the dump is performed up through double file marks or EODs.

mode is an optional parameter. If it is included, all records on the input tape, 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 size is omitted, the standard record size is used.

SEQUENCE EDITOR

The Sequence Editor routine edits EBCDIC card images by sequence number. It is more flexible than 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.

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.

To accomplish editing, the user designates columns 73 through 80 of a source card image as the "sequence field". This field consists of the ident and the sequence number.

The ident is optional and identifies a program or program segment. If defined, it begins in column 73 of the card image and is from one to six alphanumeric characters in length.

The sequence number, which is required, is the numerically sequenced part of the sequence field. It consists of two to eight decimal characters and ends in column 80. 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.

Ident and sequence number together must not total more than eight characters. Unused columns between ident and sequence number are ignored by 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

Sequence Editor performs two separate and distinct functions: it 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 Sequence Editor (SEQEDIT).

The file generation (GEN) function is used to create the permanent files initially. It is recommended that the files be sequenced as they are generated to avoid an update pass at a later stage. The user can generate either one file (terminated by an EOD from SI) wherein a single file mark is written on UO, or multiple files (terminated by two EODs from SI) wherein two file marks are written onto UO and US 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 (SEQEDIT) routine is requested via the following 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 cards are encountered from SI, or that the update function is to continue until two EOFs are encountered from UI.

The Utility Program Executive transfers control to 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 (GEN option 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 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 contiguous UI card images where the update card is to be inserted. The card image from SI and the message "INSERTED" are output on LO. Cards without sequence numbers are inserted immediately following the sequenced card preceding them. Thus, a large block of card images can be inserted by placing the proper sequence number on the first card only. The nonsequenced cards will be written on the output tape without sequence numbers. It is recommended that the tape be resequenced as it is being updated if unsequenced cards are inserted.

DELETE The DELETE command deletes one or more card images from UI. Nonsequenced cards can only be deleted by deleting from the last sequenced card preceding the nonsequenced card(s) up to and including the next sequenced card. Deleted card images are listed on LO. The form of the command is

```
                                                    73        80
!*DELETE [sequence field2]                          sequence field1
```

where

sequence $field_1$ contains the ident and/or sequence number of the first or only card image to be deleted from UI. This parameter is required.

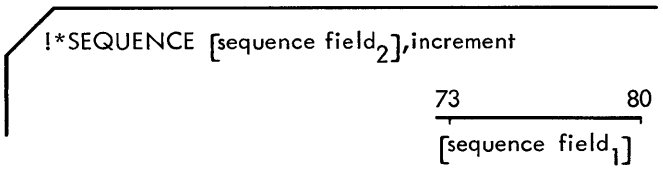
sequence $field_2$ indicates that more card images are to be deleted, from the card image specified in sequence $field_1$ up to and including the card image specified in sequence $field_2$.

SUPPRESS The SUPPRESS command is identical to the DELETE command except that deleted card images are not listed on LO. The form of the command is

```
                                                    73        80
!*SUPPRESS [sequence field2]                        sequence field1
```

SEQUENCE The SEQUENCE command is used to resequence columns 73 through 80 of the card images on UO. Only one program can be resequenced with each Sequence control

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 control 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$_1$ contains the specified card image from UI at which the SEQUENCE control command becomes effective. If omitted, the SEQUENCE control command takes effect with the next card image to be written on UO.

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$_2$ specifies 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 control command (which defines sequence fields for the input tape and update data only). If omitted, resequencing is suspended.

SEQUENCE EDITOR ERROR MESSAGES

```
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. All updating done prior to this point was written on the output tape, along with the logical end-of-tape marker. An S response causes Sequence Editor to return to RBM.

```
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 the 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" (where xxxxxxxx is the previous nonblank UI sequence field) is output on LO.

```
UNRECOV I/O UO,device
!!UKEYIN
```

An irrecoverable write error has occurred on UO. The card image and the message "UO RECORD OMITTED" or "UO FILE MARK OMITTED" are output on LO.

10. DEBUG PROGRAM

INTRODUCTION

The Debug program gives the user the capability of dumping selected portions of memory at execution time in a hexadecimal format. As do other BCM subsystems, the Debug program operates in the background under the BCM and can be loaded by the Linking Loader like any other library routine. Debug requires about 400 locations in memory. This includes the program, print buffers, etc.

The three different entries to Debug that are provided are L:DUMP, PDUMP, and DUMP. The last two are compatible with the standard FORTRAN calls to PDUMP and DUMP. Since only the hexadecimal format is provided, the FORTRAN parameters specifying the format are ignored. Output is to the DO device. If the DO device is assigned to file zero, or is not operational, no output occurs.

A call to L:DUMP is primarily used by a program coded in the Symbol language (PDUMP and DUMP entries are the standard FORTRAN calls).

CALLS TO DEBUG

The calls to Debug are

L:DUMP

```
RCPYI     P,L
B         L:DUMP
DATA      X'keys'
ADRL      L1
ADRL      L2
.
.
.
return
```

Return is to the location following the last parameter of the calling sequence. The B, X, A, E, and T registers are restored.

PDUMP

```
RCPYI     P,L
B         PDUMP
DATA      X'keys'
ADRL      L1
ADRL      L2
ADRL      L3
.
.
.
return
```

Return for PDUMP is identical to the L:DUMP entry above.

DUMP

```
RCPI      P,L
B         DUMP
DATA      X'keys'
ADRL      L1
ADRL      L2
ADRL      L3
.         .
.         .
.         .
return is to M:TERM
```

M:TERM is the BCM background termination routine.

keys is the value of the standard library argument definition keys. That is, keys is a series of two-bit codes, from left to right, that specify the addressing mode of each argument as follows:

00 means no more arguments
01 means absolute address
10 means base relative indirect
11 means base relative

One keyword can be followed by up to eight arguments. The argument order parallels the two-bit keys from left to right. As many keywords as necessary should be present.

L1 is the lower address at which to start dumping.

L2 is the upper address to terminate dumping (the last value printed is the contents of L2, in the case of an L:DUMP entry, or (L2+1), in the case of a PDUMP or DUMP entry.

If L2<L1, the two addresses will be inverted so that L2 will be the lower address and L1 the upper address.

L3 is a format control parameter of 0, and is present only for the PDUMP and DUMP entries. The L3 parameter is ignored by the Dump program, since only the hexadecimal format is available. The L3 parameter need not be present for the last triplet of parameters.

The lower address (L1) is rounded down to a multiple of 8 if output is to a Keyboard/Printer (KP), or to a multiple of 16 if output is to a Line Printer (LP). Output consists of 8 columns of data per row on the KP, or 16 columns per row on the LP. The contents of the L, T, X, B, E, and A registers are printed on each entry to the Dump routine. If there are 8 or more identical values on the KP, or 16 or more identical values on the LP (and said values are identical to the last printed value), the duplicate values are suppressed and the following message is output:

```
**LOC xxxx THRU yyyy CONTAIN zzzz**
```

A page is ejected prior to printing, and the output from different calling parameters is separated by double spacing.

The BCM Debug program uses BCM I/O routines and therefore will not execute without the BCM.

11. SYSTEM GENERATION

INTRODUCTION

System generation of a Basic Control Monitor adapted to a specific installation is performed by selection of the Monitor options required by the facility, and by definition of the installation hardware parameters, such as memory size and peripheral device numbers.

The minimum hardware requirements for BCM initialization are an ASR keyboard/printer and at least 8K of core memory.

The software required to perform initialization includes the absolute binary version of the BCM program with all options included, but without the system I/O tables defined. A self-loading bootstrap or the Stand Alone Loader is used to load the BCM program. The BCM and bootstrap modules can be in either card or paper tape format (as can system generation output).

Once the modules are loaded, the self-loading bootstrap enters a "wait" state. Then the operator must enter the device number of the keyboard/printer used by SYSGEN to output its messages and queries. When the "wait" is cleared, control is transferred to the BCM initialization and selection routines. These routines will then request input from the user, who in turn defines the selected Monitor options and hardware parameters.

The initialization and selection routines proceed to create a rebootable version of the BCM with the specific facility requirements. Any further status messages or possible error messages are output on the keyboard/printer. In the case of an error, the BCM types out a definition of the error and waits for the user to input the corrected parameters, and a Read is then retried. The system initialization need not be performed again unless changes are required in the BCM or the hardware configuration.

The rebootable version of the BCM is punched on the PB device when all input selection has been completed. This binary program (card or paper tape) is preceded by a bootstrap record that is in a special absolute format and can be loaded without any other loaders or processors. This system is the standard BCM used until there is a change in the requirements of the facility.

Each loading of the BCM causes the I/O Interrupt, Control Panel Interrupt, and BCM Control Task Interrupt to be armed and enabled.

INITIALIZATION PROCEDURE

After successfully loading the complete absolute binary BCM program, the initialization and selection routines request input via the keyboard/printer.

INPUT FORMAT

The format of all input records is free form, with the parameters beginning at the left of the record. A record is defined as one Keyboard/Printer image up to a NEW LINE (NL) character, or one EBCDIC card.

The conventions and restrictions given below must be followed in formatting input records:

1. The first blank terminates the parameter scan; therefore, comments can be included in the remaining portion of the record.
2. A backspace character (¢) will cause the previous character to be deleted (from the Keyboard/Printer).
3. Depressing the EOM key prior to the appearance of a NEW LINE character will cancel the line (on the Keyboard/Printer).
4. A hexadecimal field must begin with a plus sign.
5. A comma is used to separate fields.
6. All operational labels must begin with an alphabetical character.

INPUT PARAMETERS

Following a specific BCM request for input on the Keyboard/Printer, the user responds with one or more lines of input, as appropriate. Table 16 gives the possible BCM output messages, the user responses (parameters), and comments that define the consequences of the responses.

Table 16. Input Options and Parameters

Operation†	Function
1. Initial load of Stand-Alone Absolute Loader	1. The Stand-Alone Absolute Loader is loaded at location 8000_{10} which will then load the full BCM, as it was output from the assembler.
2. !!BCM SYSGEN	2. This is output by the BCM initialization routines as an indication that the BCM is ready to begin generating a system.

†Items in parentheses are input by the operator.

Table 16. Input Options and Parameters (cont.)

Operation[t]	Function
3. INPUT DEVICES (dtnn,dtmm)	3. The remainder of the input parameters will be read from the input device specified, where dt is the device type, and nn is the device number, in hexadecimal. The input devices are KP40 and CRnn. The output devices are KP40, LPnn or NO for Sigma 2; or KPnn, CRnn, LPnn, or NO for Sigma 3. If input is from the card reader, a summary of the information is listed on the Keyboard/Printer or on the line printer.
4. MEMORY SIZE (size)	4. Specify core size of the computer in either decimal or hexadecimal. This size must be on a 4K (K = 1024) boundary.
5. BACKGROUND START (BCM) or (address)	5. If "BCM" is specified, the initialization routines will begin the background just above all of the resident BCM, leaving no room for foreground. If protection is used, the background will begin on a page boundary. If no protection is used, the background will begin on a multiple of 16_{10}. If an address is specified, it should include space for all of the BCM options and tables selected, as well as for any foreground desired. (The size of the BCM will vary from 1970_{10} to 3500_{10}, depending on the options.) The address, if specified, must be on a page boundary. (One page equals 256_{10} words.) This must be at least one page smaller than memory size.
6. MAX. INTERRUPT LOC (address)	6. Specify the highest numbered interrupt address (264 ≤ address < 399 for Sigma 2, or 264 ≤ address ≤ 367 for Sigma 3). Example: MAX. INTERRUPT LOC 274
7. BCM INTERRUPT LOC (address)	7. Specify the interrupt address to which the BCM Control Task is connected. This may be a counter-equals-zero level or an integral or external real-time level. If it is an external level, it must be in a group with lower priority than the I/O group, or the BCM will not accept control panel interrupts. Further, the BCM interrupt level must always be the lowest priority interrupt in the system, or the real-time priorities below this BCM level will not function. Note that the highest numbered interrupt location is usually (but not always) the lowest priority interrupt.
8. BCM GROUP CODE (number)	8. Specify the group code for the BCM Control Task interrupt associated with the location above. This number must be either 0, 5, 6, ..., or X'C'. Refer to the Sigma 2 and Sigma 3 Computer Reference Manuals to determine the group code associated with the BCM interrupt.
9. INCLUDE FULL CCI (Y) or (N)	9. Specify yes (Y) or no (N). If this option is not included, the only BCM control commands that may be used are !ABS, !EOD, !FIN, !LOAD, !SLOAD, !BFORTRAN, !SYMBOL, !UTILITY, !CONCORDANCE, and !ident where ident is the name of a user's program previously loaded by the ABS control command.
10. INCLUDE PROTECT (Y) or (N)	10. Specify yes (Y) or no (N). This option is not required if there is no concurrent foreground/background processing. However, it is useful to protect the BCM from the background even without foreground. The memory protect hardware option will be required.

[t]Items in parentheses are input by the operator.

Table 16. Input Options and Parameters (cont.)

Operation[†]	Function
11. INCLUDE PARITY (Y) or (N)	11. Specify yes (Y) or no (N). The protect option without parity does not give complete foreground protection. This option requires the memory parity hardware. In a Sigma 3 configuration the inclusion of this option also includes the processing of watchdog and integral timeouts.
12. INCLUDE MULTIPLY SIM. (Y) or (N)	12. Specify yes (Y) or no (N). This option is not required if a hardware multiply capability exists. Multiply is not used by any of the standard processors except Basic FORTRAN, but is required by the math library.
13. INCLUDE DIVIDE SIM. (Y) or (N)	13. Specify yes (Y) or no (N). This option is not required if divide hardware exists. Divide is not required for any processors but is required for the math library. The divide option forces the inclusion of the multiply option as well.
14. INCLUDE M:IOEX (Y) or (N)	14. Specify yes (Y) or no (N). This option is not used by any of the processors and is needed only if the user has nonstandard peripherals or nonstandard I/O operations.
15. DEVICE FILE INFO KPnn,F (FILE #1) (dtnn,B) or (dtnn,F) (...) (-1) Example for ASR 35, with no foreground: KP40,F (FILE #1) KP40,B (FILE #2) PT40,B (FILE #3) PT40,B (FILE #4) -1	15. Specify device type name (dt) and device number (nn), in hexadecimal, for each peripheral in the system configuration. Specify whether the peripheral is to be used by the foreground only (F) or by the background only (B). The device-file number assigned to each peripheral is implicitly identical to the line number. An input of -1 terminates this step. A device-file number may be used by only one task at a time. Therefore, if several foreground tasks use the keyboard/printer, for example, each task must have a unique device-file number assigned to KPnn. Device-file number 1 is for use by the BCM Control Task for unsolicited key-ins only. A device such as an ASR 35 must have three device-file numbers if it is to be used by the background: one for the keyboard, one for the paper tape reader and one for the paper tape punch. The permissible device types to be used are Type — Device KP — Keyboard/printer PT — Paper tape LP — Line printer CR — Card reader (EBCDIC) BR — Card reader (BCD) CP — Card punch (EBCDIC) BP — Card punch (BCD) M9 — 9-track magnetic tape M7 — 7-track magnetic tape PL — Graphic plotter IP — Line printer (Model No. 7450, 225 lines per minute) IC — Card punch (Model No. 7165, 100 cards per minute) IB — Card punch (Model No. 7165, 100 cards per minute, BCD mode) XX — Nonstandard devices, to be used by M:IOEX only The keyboard/printer must always be the first device to be input, and the user must specify the device number. To distinguish the IOP type for a multi-unit device, the BCM requires either an ,I or an ,E appended to each parameter input under step 15, where ,I indicates an internal IOP and ,E indicates an external IOP. For example, M9D0,F,E indicates a magnetic tape on EIOP.

[†]Items in parentheses are input by the operator.

Table 16. Input Options and Parameters (cont.)

Operation[t]	Function
16. BACKGROUND OP LBL (op lbl = device-file number) or (device unit number = device-file number) (...) (-1)	16. Assign the permanent background operational labels for the system. Use the device-file numbers defined in step 15. An assignment of "0=" will reserve space in the operational table to allow nonstandard FORTRAN device unit number to be assigned at execution time, if the full CCI is included. (If the full CCI is not included, all FORTRAN device unit numbers that are to be referenced must be defined here.) A permanent assignment to file zero means that this label is not to be used. (Example: LL=0.) If a label has been previously defined, the new value overrides the old. An input of -1 will terminate this step. The standard operational labels are Label — Reference OC — Operator's Console SI — Symbolic Input AI — Absolute Input BI — Binary Input LI — Library Input LO — Listing Output LL — Listing Log DO — Diagnostic Output BO — Binary Output PB — Punch BCM CC — Control Command Input X1 — Intermediate Scratch UI — Utility Input UO — Utility Output The standard FORTRAN device unit numbers are 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
17. FOREGROUND OP LBL (op lbl = device-file number) or (device unit number = device-file number) (...) (-1)	17. Assign foreground labels, using the same method as for the background. There are no standard foreground labels (it is not actually necessary to specify any foreground labels). This step is also terminated by a -1.
18. BCM ENDS AT LOC xxxx	18. At this point, BCM initialization is complete and the rebootable version of the user's BCM has been output on the PB device, if system initialization was successful. If initialization was not successful, the BCM will output the message !!BCM OVERLAPS BACKGND, and the initialization procedure must be restarted from the beginning.

[t]Items in parentheses are input by the operator.

Table 16. Input Options and Parameters (cont.)

Operation†	Function
19. BACKGROUND BEGINS AT LOC xxxx	19. If the "BCM" format was specified in step 5, above, this message is output to inform the operator of the actual beginning of background.
20. !!SET PARITY TO 'INTERRUPT'	20. This indicates that the BCM includes the parity option, and the operator should set the specified condition on the control panel.
21. !!AFTER 'WAIT', SET PROTECT 'ON'	21. This indicates that the BCM includes the protect option. The operator must not set the protect to ON until the BCM enters the "wait" state.
22. !!INTERRUPT AND KEY-IN AN 'S' TO BEGIN	22. This indicates that the BCM is in core storage and ready for execution. This message is also typed after each successful loading of the reboot-able BCM program.

†Items in parentheses are input by the operator.

ERROR MESSAGES

During initialization, invalid input causes one or more of the following error messages to be typed out on the Keyboard/Printer:

Message	Meaning	Recovery
!!INVALID PARAMETER	Input parameter is out of expected range	Correct input and retry this command
!!FORMAT ERROR	Input format not valid	Check format, retry the command that caused the failure
!!BCM OVERLAPS BACKGND	Error in size specifications	Check sizes, restart from beginning

BACKGROUND PROCESSORS

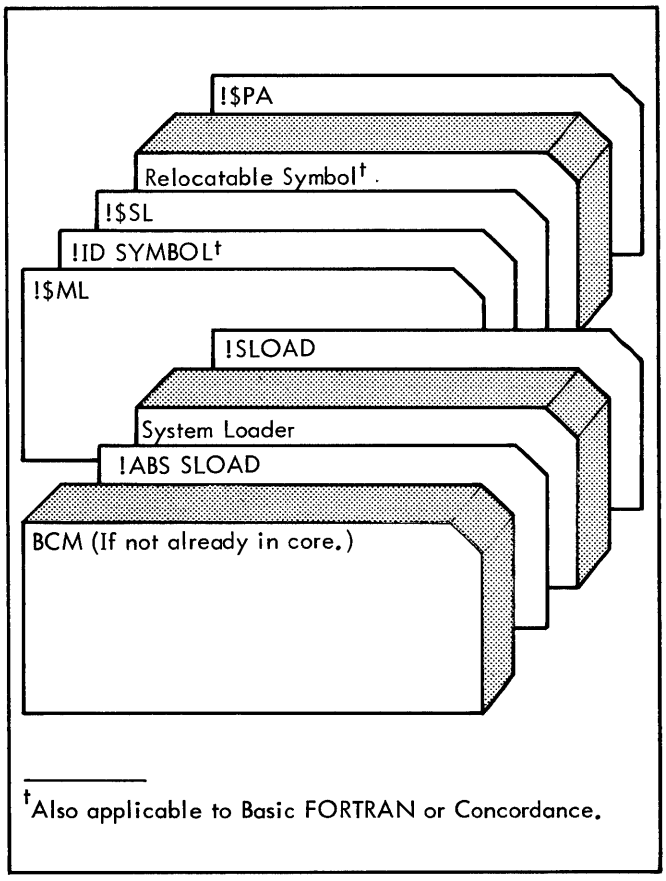
Under the BCM, background processors (with the exception of the Linking Loader and System Loader; see below) are supplied to the user in relocatable format as either paper tapes or cards. The processors are loaded into core storage at the position they will occupy at execution time. The System Loader then produces absolute object modules that are loaded later via the ABS control command prior to execution.

The generation of absolute load modules is mandatory only for the utility subsystem. All other relocatable and absolute processors are compatible with the ABS Loader. (For minimum core configurations, FORTRAN requires absolute object module generation.)

Only one system loading process is necessary, unless changes to the size limits of background are required.

The System Loader and the Linking Loader are in absolute binary and use the special addressing features of Sigma 2/3 hardware to relocate themselves, regardless of the location of the background space.

Sample deck structures to generate absolute background processors are shown below. These deck structures and control commands assume that the processor is being generated in the machine on which it is to be run; i.e., the load location is set by the Monitor background plus 20_{16} rather than being specified on an $SL control command.



†Also applicable to Basic FORTRAN or Concordance.

Figure 12. Deck Setup to Punch Absolute Background Processor

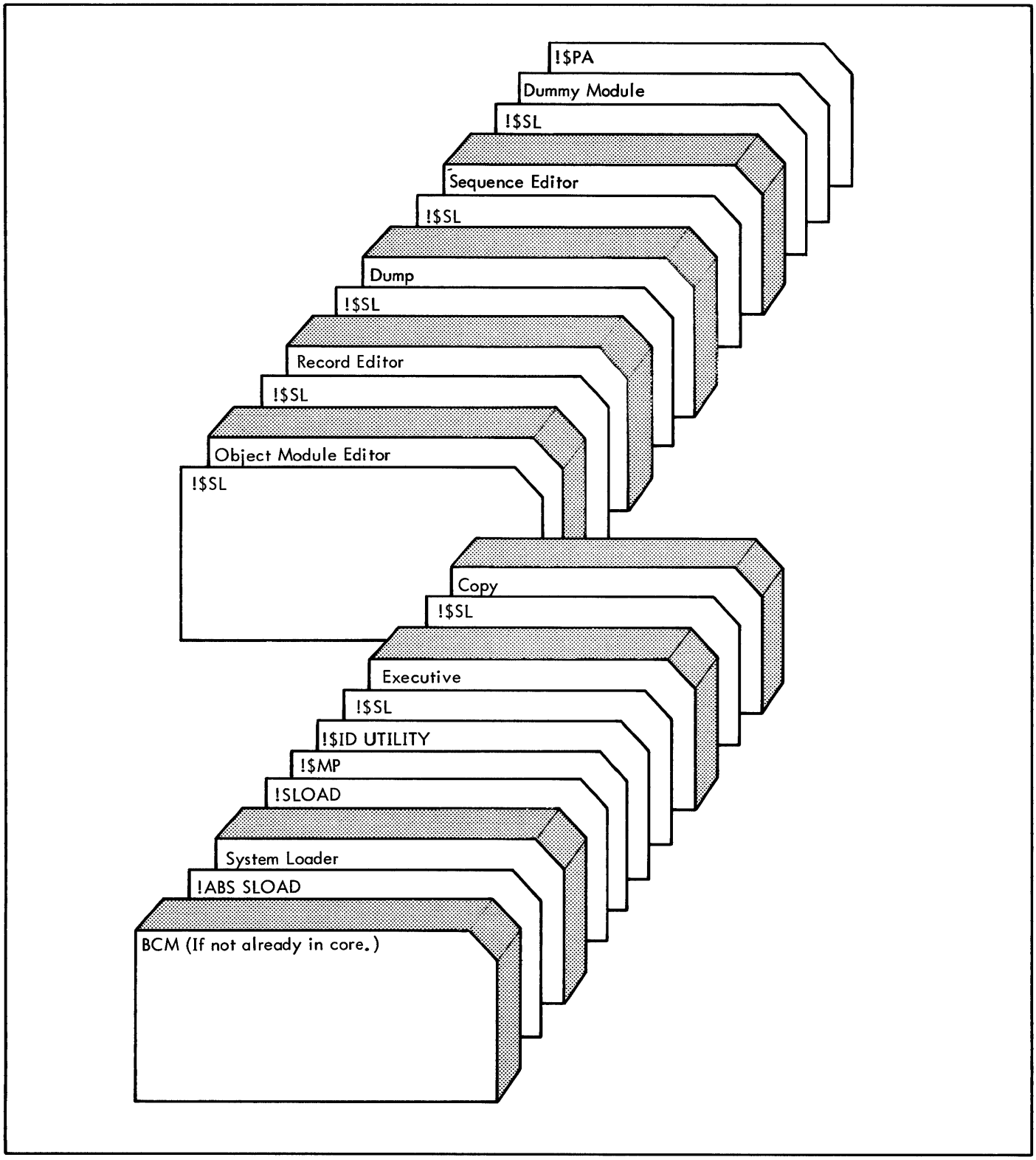


Figure 13. Deck Setup to Punch Absolute Utility Package

APPENDIX A. SIGMA 2/3 STANDARD OBJECT LANGUAGE

INTRODUCTION

The SDS 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 SDS Sigma 2/3 Linking Loader and System 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 Linking Loader has the facility to load and link several object modules together to form an executable program.

The Sigma 2/3 BCM 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 BCM Absolute Loader, Linking Loader, and System 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.

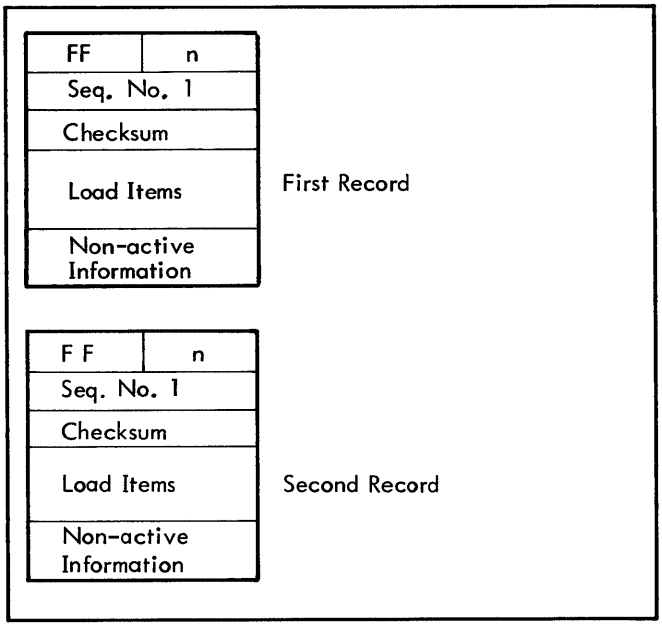


Figure A-1. Typical Object Module of M Records

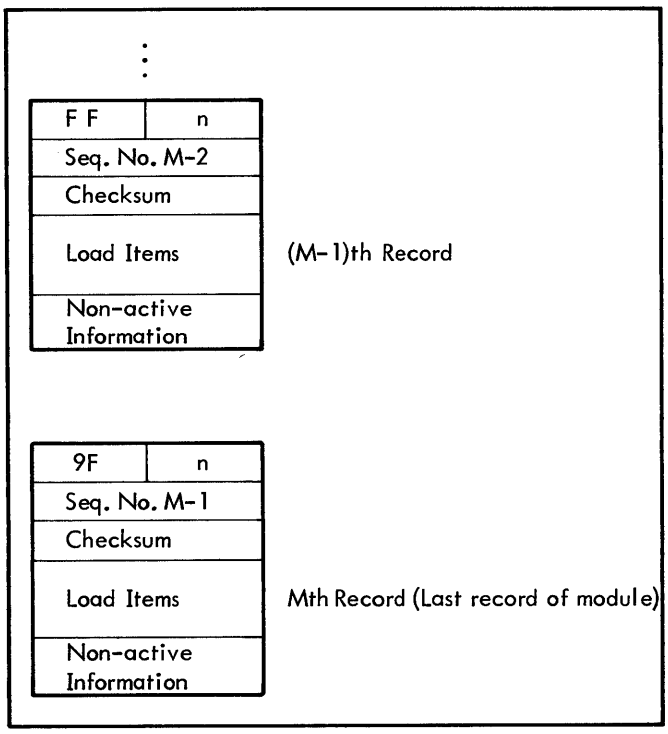
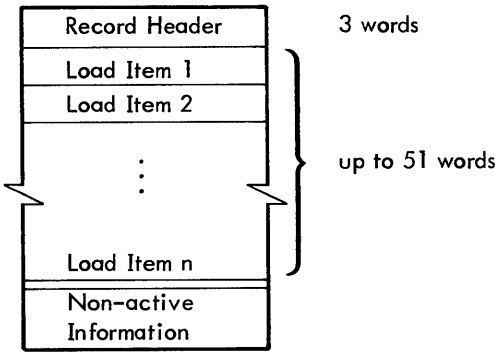


Figure A-1. Typical Object Module of M Records (cont.)

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 Non-active Information in the following arrangement. The Record Header and Load Items are considered the "active" portion of the record.

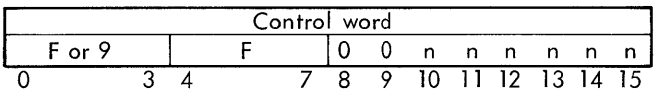


The "active" portion of the record is that information concerning type, sequence number, checksum and binary data usually processed by loaders. The "non-active" portion may contain sequence or identification information, or it may be empty. It is not processed by the loaders.

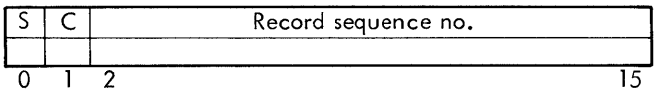
FORMAT OF RECORD HEADER

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.

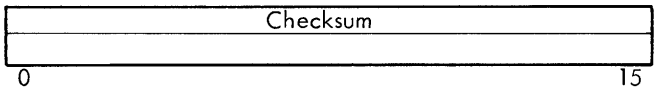
word 0



word 1



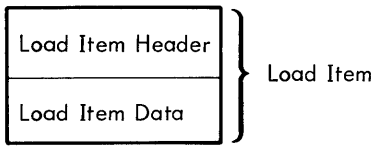
word 2



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

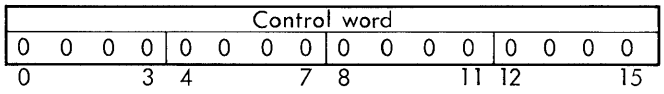
bits 8-15 Number of data words in the load item (excluding item header).

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)

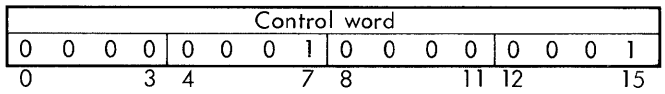
word 0



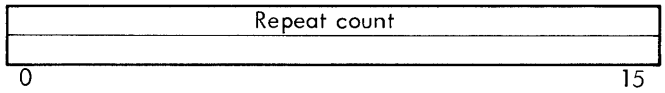
There is no body of data. Padding words are ignored by the loader. The object language allows padding as a convenience for processors.

REPEAT LOAD (Type 0, Subtype 1)

word 0



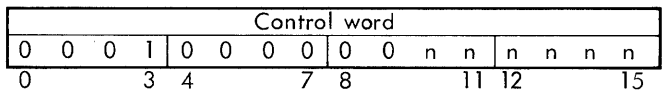
word 1



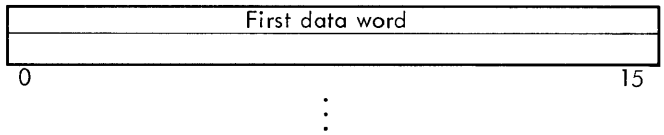
This item repeats the next load item a specified number of times. The load item (type 1, 2, or 3 only) immediately following the repeat load is repeated (i.e., loaded) in its entirety the number of times indicated by the data word.

UNRELOCATED LOAD (Type 1)

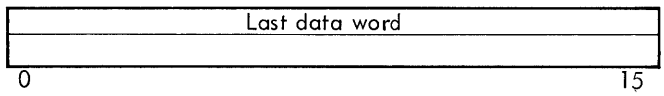
word 0



word 1



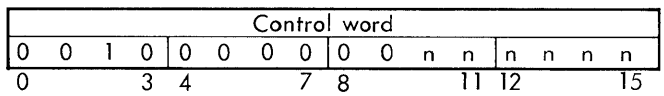
word n



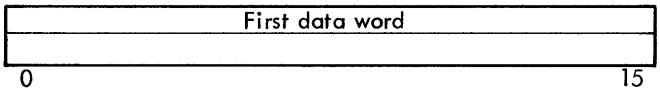
This item loads n words without relocation.

RELOCATED LOAD-MODULE BASE (Type 2)

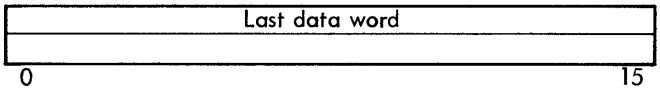
word 0



word 1



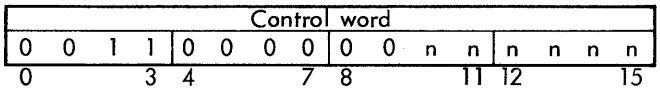
word n



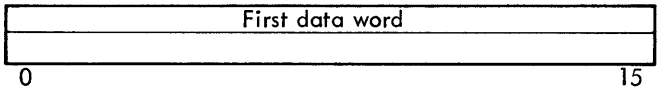
This item loads n words with module relocation. The relocation bias of the current object module is added to each data word in the item.

RELOCATED LOAD-COMMON BASE (Type 3)

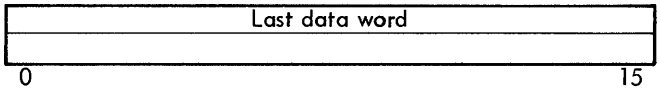
word 0



word 1



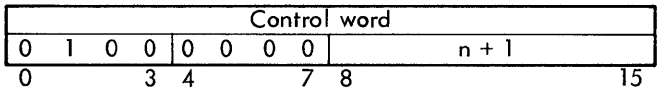
word n



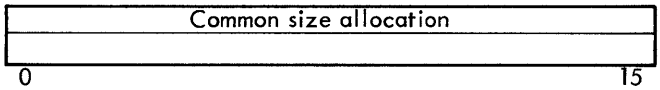
This item loads n words with a common base relocation.

START MODULE (Type 4)

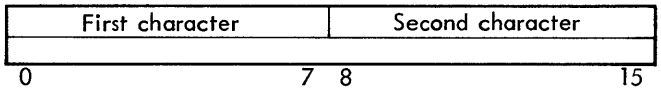
word 0



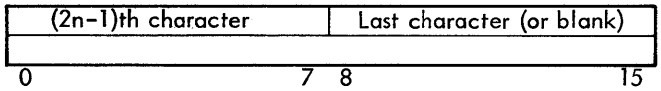
word 1



word 2



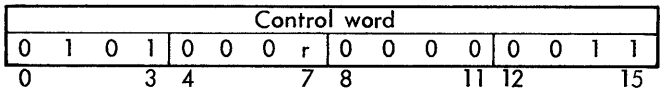
word n + 1



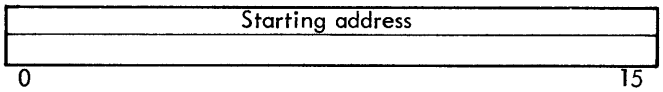
This item identifies the start of the object module. The characters in words 2 through n + 1 are the program name (identification) for the module.

END MODULE (Type 5)

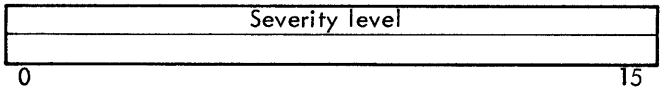
word 0



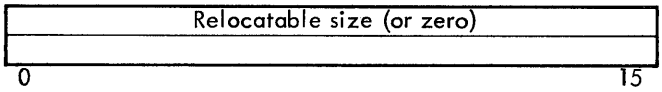
word 1



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.

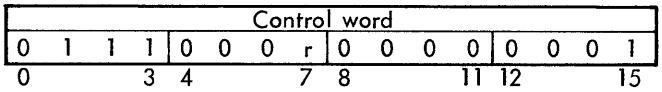
The severity level in word 2 is defined as the highest level reached during processing.

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.

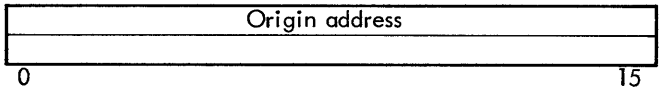
A starting address of absolute 0 indicates there is no starting address for this module.

LOAD ORIGIN (Type 7)

word 0



word 1



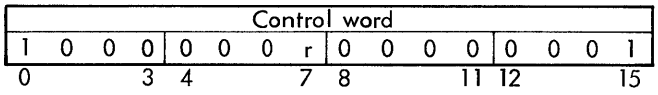
This item sets the origin within the object module. In the control word (word 0), the origin is defined in bit 7

where

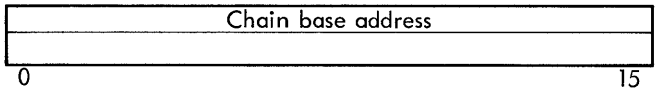
r = 0 indicates relocatable origin.
r = 1 indicates absolute origin.

RELATIVE LOCATION POINTER (Type 8)

word 0



word 1



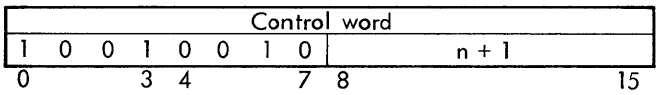
This item establishes the chain base for later chain resolution. In the control word (word 0), the chain base address is defined in bit 7

where

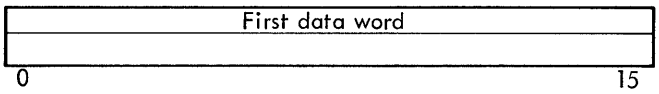
r = 0 indicates a relocatable address.
r = 1 indicates an absolute address.

NAME DEFINITION (Type 9)

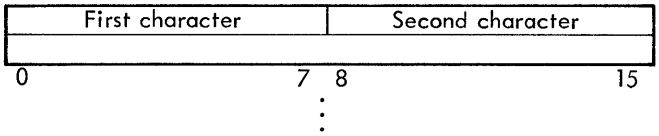
word 0



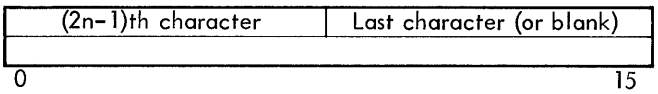
word 1



word 2



word n + 1

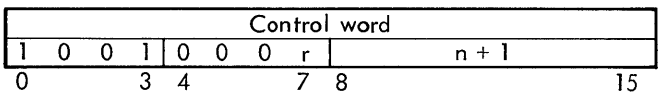


This item identifies a name as a definition within the object module.

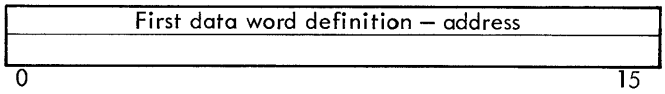
All name definitions immediately follow the start-module item and must precede all other load items. For each name definition, an address definition should appear later in the object module.

ADDRESS DEFINITION (Type 9)

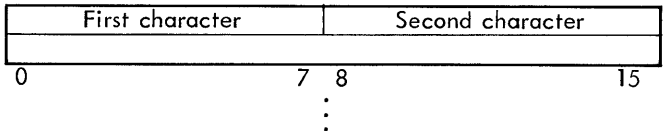
word 0



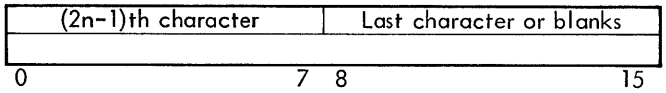
word 1



word 2



word n + 1



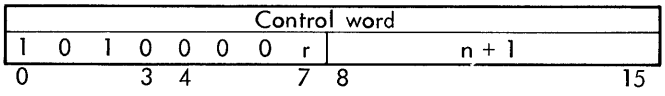
This item associates a location in the module with a definition name (characters in words 2 through n + 1) for other modules to reference. In the control word (word 0), the definition address is defined in bit 7

where

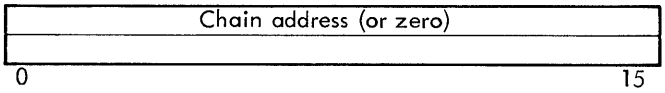
r = 0 indicates relocatable definition address.
r = 1 indicates absolute definition address.

EXTERNAL REFERENCE (Type A)

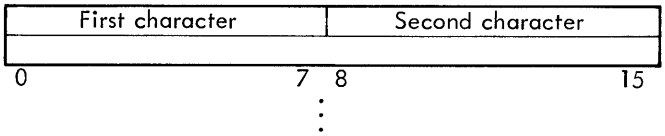
word 0



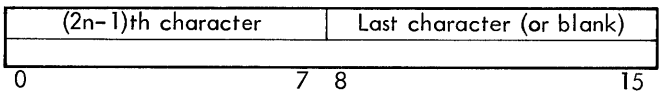
word 1



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

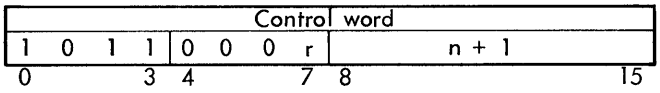
where

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

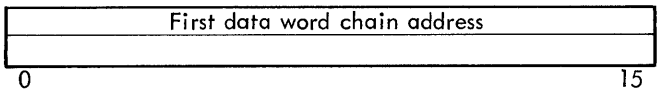
Note: If there is no chain address, the reference address is zero and is used for library searching purposes only.

SECONDARY REFERENCE (Type B)

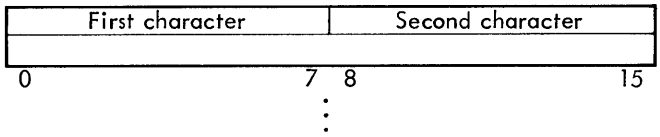
word 0



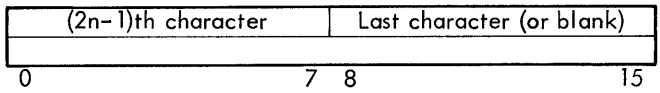
word 1



word 2



word n + 1



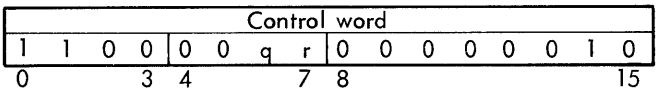
This item states a name (characters in words 2 through n+1), defined in another module, whose address may be inserted in a chain of locations within the module. This item is identical to type A, above, except that it does not force loading of the routine from the library. In the control word, the chain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

where

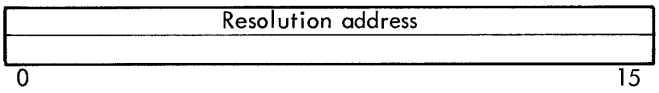
r = 0 indicates a relocatable chain address.
r = 1 indicates an absolute chain address.

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

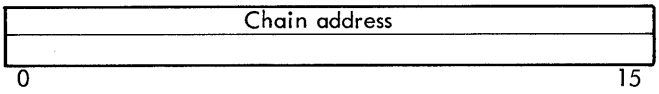
word 0



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.

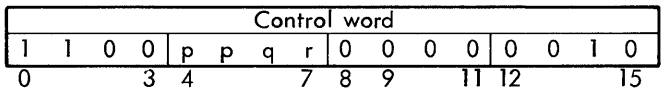
An address literal chain is a threaded list of forward references to a single location in a program. The definition value (called the resolution address) can be output as an address literal chain resolution (Type C, subtypes 0, 1, 2, and 3). The chain address points to the beginning of the threaded list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

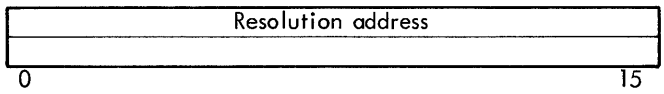
Note that external reference (REF) (type A) and secondary reference (SREF) (type B) chains are structured in the same manner, but resolved by the loader using an external definition value (type 9).

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

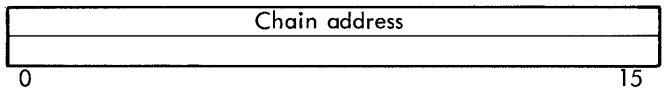
word 0



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 1-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. STANDARD BCM ABORT CODES

Code	Meaning
OP	Operator abort, from unsolicited key-in.
PV	Protection violation.
PE	Parity error in background (perhaps attempting to read from unavailable memory).
IO	Irrecoverable I/O error.
AE	Assignment error during loading; improper I/O assignment or invalid format.
CC	Error in control cards or in sequence of job stack.
SQ	Sequence error in absolute binary deck.
CS	Checksum error from card or paper tape input.
XE	Invalid transfer address, fatal error in loading, or improper name for background program.
SI	Irrecoverable input error on SI device.
BI	Irrecoverable input error on BI device.
LI	Irrecoverable input error on LI device.
LO	Irrecoverable output error on LO device.
BO	Irrecoverable output error on BO device.
MD	Multiply or divide instruction without supporting hardware or software.
TY	Program being loaded with !ABS command contains an external or relocatable load item.

Note: The processing of the job stack is discontinued after any abort message, to allow the operator to correct the condition. The Monitor will continue to read from CC when a key-in of S is given, and will attempt to recover if the new commands are for the current job. If a new job is input, the current job is overwritten in core storage.

INDEX

D

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

SDS

Scientific Data Systems A XEROX COMPANY
701 South Aviation Blvd. / El Segundo, California 90245 (213) 772-4511 / Cable SCIDATA / Telex 674839 / TWX 910-325-6908

EASTERN TECHNOLOGY CENTER
12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

PRINTED CIRCUITS DEPT.
600 East Bonita Avenue
Pomona, Calif. 91767
(714) 624-8011

TECHNICAL TRAINING
5250 West Century Blvd.
Los Angeles, Calif. 90045
(213) 772-4511

INTERNATIONAL MANUFACTURING SUBSIDIARY

Scientific Data Systems Israel, Ltd.
P.O. Box 5101
Haifa, Israel
04-530253, 04-64589
Telex 922 4474

SALES OFFICES

Western Region

5045 N. 12th St.
Phoenix, Arizona 85014
(602) 264-9324

1360 So. Anaheim Blvd.
Anaheim, Calif. 92805
(714) 774-0461

*5250 West Century Blvd.
Los Angeles, Calif. 90045
(213) 772-4511

Vista Del Lago Office Center
122 Saratoga Avenue
Santa Clara, Calif. 95050
(408) 246-8330

3333 South Bannock
Suite 400
Englewood, Colo. 80110
(303) 761-2645

Fountain Professional Bldg.
9004 Menaul Blvd., N.E.
Albuquerque, N.M. 87112
(505) 298-7683

*Regional Headquarters

El Paso Natural Gas Bldg.
Suite 201
315 E. 2nd South Street
Salt Lake City, Utah 84111
(801) 322-0501

400 Bldg., Suite 415
400 108th Avenue, N.E.
Bellevue, Wash. 98004
(206) 454-3991

Midwestern Region

*2720 Des Plaines Avenue
Des Plaines, Illinois 60018
(312) 824-8147

Clausen Bldg., Suite 310
16000 W. Nine Mile Road
Southfield, Michigan 48124
(313) 353-7360

4367 Woodson Road
St. Louis, Missouri 63134
(314) 423-6200

Seven Parkway Center
Suite 238
Pittsburgh, Pa. 15220
(412) 921-3640

Southern Region

State National Bank Bldg.
Suite 620
200 W. Court Square
Huntsville, Alabama 35801
(205) 539-5131

Orlando Executive Center
1080 Woodcock Road
Orlando, Florida 32803
(305) 841-6371

2964 Peachtree Road, N.W.
Suite 350
Atlanta, Georgia 30305
(404) 261-5323

Jefferson Bank Bldg.
Suite 720
3525 N. Causeway Blvd.
Metairie, Louisiana 70002
(504) 837-1515

4900 S. Lewis Avenue
Tulsa, Oklahoma 74105
(918) 743-7753

8383 Stemmons Freeway
Suite 233
Dallas, Texas 75247
(214) 637-4340

*2300 West Loop South
Suite 150
Houston, Texas 77027
(713) 623-0510

Eastern Region

20 Walnut Street
Wellesley Hills, Mass. 02181
(617) 237-2300

Brearley Office Building
190 Moore Street
Hackensack, N. J. 07601
(201) 489-0100

280 North Central Avenue
Hartsdale, New York 10530
(914) 948-2929

*1301 Avenue of the Americas
New York City, N.Y. 10019
(212) 765-1230

673 Panorama Trail West
Rochester, New York 14625
(716) 586-1500

P.O. Box 168
535 Pennsylvania Ave.
Ft. Washington Industrial Park
Ft. Washington, Pa. 19034
(215) 643-2130

Washington (D.C.) Operations

*2351 Research Blvd.
Rockville, Maryland 20850
(301) 948-8190

Canada

864 Lady Ellen Place
Ottawa 3, Ontario
(613) 722-8387

Oil Exchange Building
309 7th Avenue, S.W.
Calgary 2, Alberta
(403) 265-8134

280 Belfield Road
Rexdale 605, Ontario
(416) 677-8422

INTERNATIONAL OFFICES & REPRESENTATIVES

European/African Headquarters

Scientific Data Systems
I.L.I. House, Olympic Way
Wembley Park (London)
Middlesex, England
(01) 903-2511, Telex 27992

Sweden

Nordisk Elektronik AB
Stureplan 3
Stockholm 7
(08) 24 83 40

Denmark

A/S Nordisk Elektronik
Danasvej 2
Copenhagen V
EVA 8285/EVA 8238

Norway

Nordisk Elektronik (Norge) A/S
Middelthunsgt. 27
Oslo 3
(2) 60 25 90

France

Compagnie Internationale pour l'Informatique, C.I.I.

Executive and Sales Offices
66, Route de Versailles
78-Louveciennes
Yvelines
951 86 00 (Paris area)

Manufacturing and Engineering
Rue Jean Jaures
78-Les Clayes-sous-Bois
Yvelines

Israel

Elbit Computers Ltd.
Subsidiary of Elron Electronic Industries Ltd.
88 Hagiborim Street
Haifa
6 4613

90 10 64C