96769400

CONTROL DATA CORPORATION

MASS STORAGE OPERATING SYSTEM (MSOS) VERSION 5 REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:

CYBER 18 MODELS 17/20/30 TIMESHARE

1700

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(1/77)	
B	Manual revised to complete ASCII Conversion Table. The affected pages are: title page, ii, iii/iv, G-3, G-4,
(2/77)	and comment sheet.
C	Manual revised to contain new information on peripheral equipment; 1860 LCTT/FORMATTER, 1827-2 Line
(10/77)	Printer, 1811-2 CDT, 1843-2 CLA, MSOS 5 Console. Incorporates PSRs 3658, 4245, 4310, 4366, 4375, 4418,
	and 4421.

Publication No.
96769400

REVISION LETTERS I, O, Q AND X ARE NOT USED

Printed in the United States of America

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4455 Eastgate Mall
La Jolla, California 92037

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Cover	—
Title page	—
ii	C
iii/iv	C
v/vi	A
vii thru x	C
1-1 thru 1-12	A
1-13	C
1-14	C
1-15 thru 1-19	A
2-1 thru 1-6	A
2-7	C
2-8	C
2-9 thru 2-19	A
3-1	A
3-2	C
3-3thru 3-8	A
3-9	C
3-11	A
3-12	A
3-13	A
3-14	C
3-15	C
3-16	C
3-17	A
3-18	A
3-19	C
3-20	A
3-21	A
4-1	A
5-1 thru 5-4	A
6-1 thru 6-4	A
6-5	C
6-6	A
6-7	A
7-1 thru 7-5	A
8-1	A
8-2	A
9-1	C
9-2 thru 9-8	A
10-1 thru 10-20	A
11-1	C
11-3 thru 11-12	A
12-1 thru 12-5	A
12-6	C
13-1 thru 13-8	A
14-1	C
14-2 thru 14-32	A
14-33	C
14-34 thru 14-36	A
14-37	C
14-38	C
14-39	A
14-40	C
14-41	C
14-42	A
14-43	C
14-44	C
14-44.1	C
14-45	C
14-46 thru 14-49	A
14-50	C
14-51	C
14-52	A
14-53	A
14-54	C
14-55	A
A-1 thru A-9	A
B-1 thru B-4	A
C-1 thru C-3	A
C-4	C
C-5	A
C-6	A
C-7	C
C-8	A
D-1	A
D-2	A
E-1	A
F-1	A
F-2	C
F-3 thru F-5	A
G-1,	A
G-2	A
G-3	B
G-4	B
H-1 thru H-5	A
I-1	A
J-1	A
J-2	C
J-3	A
J-4	C
J-5	A
J-6	C
J-7	A
J-8	C
K-1 thru K-12	A
L-1 thru L-3	A
Index-1	A
Index-2	C
Index-3 thru Index-7	A
Comment sheet	C
Mailer	—
Cover	—

PAGE	REV

PAGE	REV

PAGE	REV

PAGE	REV

PREFACE

This manual is directed at the CDC® CYBER 18/1700 MSOS Version 5 programmer and assumes a basic knowledge of the system.

This manual uses 1700 when referencing features on 1704, 1714, 1774, and 1784 (CYBER 18-17) systems and CYBER 18 when referencing features on 18-20 or 18-30 Timeshare systems.

All numbers in this manual are assumed to be decimal unless otherwise specified.

Additional information may be found in the following publications:

Publication	Publication Number
MSOS 5 Installation Handbook	96769410
MSOS 5 Diagnostic Handbook	96769550
MSOS 5 Macro Assembler Reference Manual	60361900
MSOS 5 Instant	96769430
MSOS 5 File Manager Version 1 Reference Manual	39520600
CYBER 18/1700 Peripheral Drivers Reference Manual	96769390
Small Computer Maintenance Monitor Reference Manual	39520200
MS FORTRAN Version 3A/B Reference Manual	39518900
RPG II Version 2 Reference Manual	96769010
AUTRAN 2 Reference Manual	96729800
Sort/Merge Reference Manual	96769260
CYBER 18 Processor with Core Memory (Macro Level) Reference Manual	88973500

This product is intended for use only as described in this document. Control Data Corporation cannot be held responsible for the proper functioning of undescribed features or undefined parameters.

CONTENTS

The CDC® 1704, 1714, 1774, 1784 (CYBER 18-17), 18-20, and 18-30 Timeshare Computer Systems are small, high-speed digital processors that are designed to perform a variety of on-line control, real-time data acquisition, business, and batch job processing applications. The Mass Storage Operating System Version 5 (MSOS 5) is provided to manage the system's resources on a priority basis within a multiprogramming environment. MSOS queues requests for input/output data transmission and program execution by priority level. The program task that is selected for execution is the one with the highest priority level.

A hardware interrupt system provides concurrent I/O operations and computation.

A main memory is divided into two functional entities: protected and unprotected memory. Protected memory (the foreground) is reserved for executing all parts of the operating monitor and the user's on-line high-priority application programs. Unprotected memory (the background) is used for execution of batch job processing and program checkout at low priority levels. MSOS can swap the contents of unprotected memory to mass memory to make that space available to foreground programs. When higher priority foreground tasks are completed, the swap is reversed, allowing the background program to continue its processing.

MSOS is modular in design, providing the user with considerable flexibility to perform on-line system modification.

The operating system is highly modularized with both necessary and optional portions. Among the necessary functions are:

- System monitoring
- I/O control
- Background processing control
- Mass storage library management
- Major utilities

Among the optional functions are:

- File management
- Debugging aids
- Other utilities

MSOS also provides control for important systems that are separate software systems. Among the most frequently used systems are as follows.

- Mass-storage-resident FORTRAN compiler
- Macro assembler for 1700 language
- Report generator (RPG II)
- Sort/merge

FEATURES

LANGUAGES

The basic MSOS features are summarized below. The software elements which implement these features are shown in table 1-1.

- MSOS uses the basic 1700 assembly language and supports the CYBER 18 enhanced instructions. With the macro assembler present, user programs may be written using standard macro commands as well as special macro commands defined to fit the user's special needs.
- If the FORTRAN compiler is present, user programs may be written using the FORTRAN instruction set.

FOREGROUND PROCESSING

- A series of request processors allow programs to link to one another and to communicate with I/O drivers that move data to/from computer peripherals. Background program linkage to foreground programs is strictly controlled by a protect bit system. This minimizes the possibility of a background program destroying a foreground program or meddling with foreground (especially process control) data.
- Monitor and request processors are re-entrant.
- Monitor schedules programs and oversees core allocation. Scheduled tasks are queued for execution on a priority basis.
- Space allocation and loading provide main memory for execution on a priority basis. If necessary, background programs are swapped out of main memory so foreground programs have processing space.

For foreground programs, one system of priorities determines the program's place in the queue of programs awaiting execution, and a related priority determines the amount of main memory available for loading a mass-storage resident program for execution.

TABLE 1-1. MSOS ELEMENTS AND AIDS

Major System Components (Foreground – Required)	Languages/Aids
Monitor (includes scheduling, dispatching, core allocation, and common data) I/O drivers (core or mass-storage resident) Initializer (leaves core after loading) User applications programs Job processor (operates at minimum foreground priority, but the background programs operate below minimum foreground priority) Relocatable binary loader – optional File manager Utilities Re-entrant FORTRAN routine	Languages 1700 Assembly MS FORTRAN Aids Macro assembler Library build and edit: COSY Utilities Debugging aids (including ODEBUG) Small Computer Maintenance Monitor (SCMM) Breakpoint/recovery Report generator

- CYBER 18 extended core capability allows use of main memory above 65K as if it were a mass storage peripheral device. Below 65K, core is fully executable by word addressing.
- Initialization and checkout program are provided. Autoload capability is available after initialization is completed.

I/O PROCESSING SUBSET OF FOREGROUND OPERATIONS

- Interrupt handlers process the 16 separate hardware interrupts. Assignment of system priorities can give preference to I/O interrupt processing over all other programs. Interrupted program parameters are saved and then are restored after the I/O data transfer routines service the hardware requests. Also, each interrupt is assigned a priority so that the more time-critical I/O devices are processed in preference to those whose data transfer can remain on the I/O lines for a longer period (e.g., the card reader is serviced in preference to the disk).
- Each I/O device has its own driver. Maximum I/O error detection and recovery procedures can be included in each driver. (Kernel drivers usually allow the user to select the level of recovery procedure desired. This is done at customization time.)
- Alternate device handling techniques allow data for a nonoperable device to be transferred automatically to an operable device (e.g., a message output to the listing device which is down can be output on the comment device).
- Unrecoverable I/O errors are logged in an engineering file.
- 1700 systems: I/O channels operate in one of three ways:
  - -A and Q registers are used to transfer data one word at a time.
  - -Direct storage access (DSA) provides I/O-to-main-memory transfers for mass storage devices.
  - -The 1706 Buffered Data Channel allocates DSA access to I/O devices which would otherwise not have a DSA capability
- CYBER 18 systems: I/O channels also operate in one of three ways:
  - -A and Q registers are used to transfer one word at a time.
  - -Direct memory access (DMA, the functional equivalent of DSA for 1700 systems) provides I/O-to-main-memory transfers.

- Auto data transfer (ADT) provides A/Q transfers on a buffered basis, relieving the requesting module of the burden of transferring data one word at a time.

- If SCMM (small computer maintenance monitor) is included, an on-line I/O diagnostic capability is present.

BACKGROUND PROCESSING

- A job processor provides background (batch) program processing.

- If foreground tasks need space currently allocated to background, unprotected core is written to mass storage (swapped) so the background program space can be reallocated to the foreground task.

- A relocatable binary loader absolutizes programs as they are read from the program library or peripheral device. The loader links them to all entry points in other programs required to complete the overall task.

- CYBER 18 bounds registers are provided for background limits.

PROGRAM AND DATA MAINTENANCE

- On-line and off-line debugging aids are provided. These include breakpoint and recovery capability.

- On-line system modification is provided.

- Extensive system maintenance and utility routines allow easy altering of mass storage programs in the two libraries. A system library holds foreground programs and a program library holds background programs.

- With the COSY data compression routine present, program maintenance uses condensed Hollerith code to minimize card deck (or other storage medium) size and to speed and simplify program maintenance.

- If the file manager is present, the system can create and maintain sequential and indexed files.

- If the report generator is present, the system can tailor reports from the filed data.

- If sort/merge is present, the system can sort files by keys and merge files selectively to update them.

- Text editing is available for background files.

- CYBER 18 extended core storage can be used to provide mass-storage compatible data regions.

OTHER FEATURES

- Date and time-of-day routines are available.

- A floating point arithmetic option (hardware or software) is available.

COMPUTER

The CDC® 1704, 1714, 1774, 1784 (CYBER 18-17), 18-20, and 18-30 Timeshare Computer Systems run under MSOS. These are stored program digital computers designed for high computation and I/O speed. The program protect features and high reliability under a wide range of environmental conditions make it suitable for real-time, on-line, business, or control applications.

The interface of the computer system is capable of accepting a variety of peripheral devices.

The basic computer system for the 1704, 1714, 1774, and 1784 (18-17) provides high-speed, random-access central memory storage for 4096 18-bit (sixteen bits and one parity and one protect bit) words. The storage capacity of the 1704/1774 may be expanded to 32K in 4K increments; the 1714 may be expanded from 24K to 65K in 8K increments; and the 1784 (18-17) may be expanded to 65K in 4K increments.

The CYBER 18-20 is a micro-programmable processor that accommodates from 32,768 to 262, 144 eight-bit bytes (one protect and one parity bit is provided for each two bytes) of MOS main memory with an effective read/write cycle time of 750 nanoseconds. The 18-20 emulates the basic as well as the enhanced 1700 instruction set.

The CYBER 18-30 Timeshare System consists of dual micro-programmable processors, each accommodating from 32,768 to 262,144 eight-bit bytes (one protect and one parity bit for each two bytes) of MOS main memory with an effective read/write cycle time of 750 nanoseconds. Each processor can address the memory of the other for a total capability of 524,288 eight-bit bytes of main memory. The processors emulate the basic as well as the enhanced 1700 instruction set. The CYBER 18-30 Timeshare System runs under a modified MSOS.

The program protect system makes it possible to protect a program in the computer from any other unprotected program.

Figure 1-1 shows the protected and unprotected programs and indicates how unprotected, product set, and user programs are related.

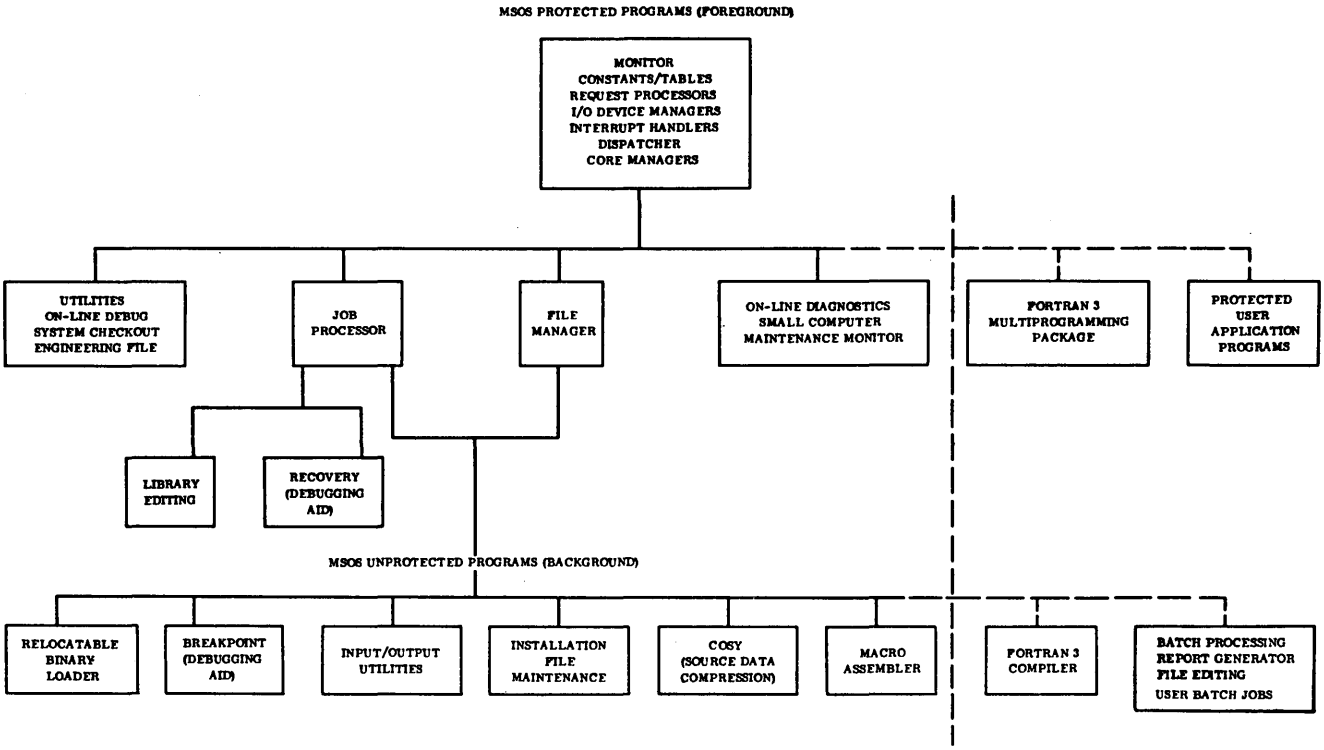


Figure 1-1. Diagram of Protected/Unprotected/Product Set/User Programs

HARDWARE CONFIGURATIONS

To achieve standardization of equipment numbers, interrupt lines, and logical units, MSOS supports only one like function peripheral device in a given system. Other configurations are possible but require deviation from the standards. Figure 1-2 is a typical configuration.

MINIMUM CONFIGURATION

The following are the MSOS minimum machine configurations.

Typical 1704 Configuration

1704 Computer with 4096 words of memory

1708 Storage Increment with 4096 words of memory (3)

1711-2 Teletypewriter, Comment Device

1777 Paper Tape Station, Input/Output Device

1739-1 Cartridge Disk/Library Device

1705 Interrupt/Data Channel

Typical 1714 Configuration

1714 Computer with 24,567 words of memory

1711-2 Teletypewriter, Comment Device

1777 Paper Tape Station, Input/Output Device

1739-1 Cartridge Disk Library Device

Typical 1774 Configuration

1774 System Controller with 4096 words of memory

1772 Memory Module with 4096 words of memory (3)

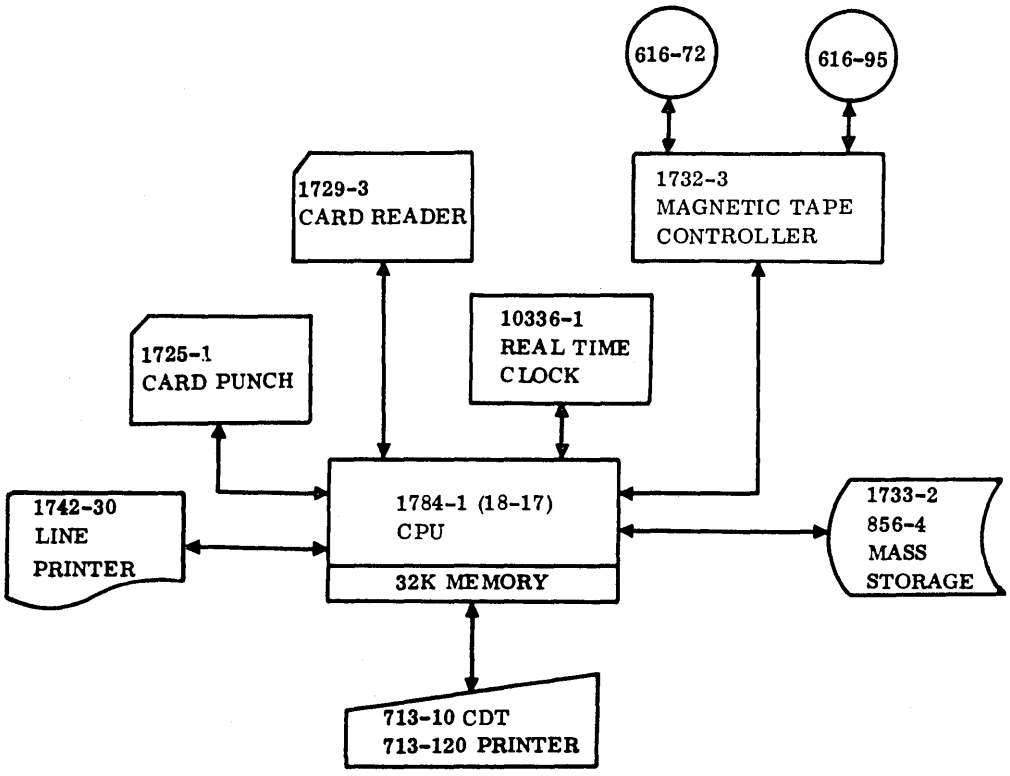
1773 Direct Storage Access

1775 Interrupt/Data Channel

1711-3 Teletypewriter, Comment Device

1777 Paper Tape Station, Input/Output Device

1739-1 Cartridge Disk, Library Device



NOTES:
1. CONFIGURATION IS TYPICAL, BUT NOT THE MINIMUM; NOR IS IT ALL INCLUSIVE AS TO EACH SUPPORTED DEVICE.
2. MORE DISKS, READERS, TAPES, ETC., MAY BE INCORPORATED.
3. MEMORY CONSISTS OF COMBINATIONS OF MODULES DEPENDENT ON CPU AND CORE SIZE.

Figure 1-2. Typical Equipment Configuration

Typical 1784 (CYBER 18-17) Configuration

1784-1 Computer (900-nanosecond cycle time) with 4096 words of memory

1782-1 Memory Module (900-nanosecond cycle time) with 4096 words of memory (3)

1711-4 Teletypewriter, Comment Device

1732-3 1 x 4 Magnetic Tape Controller, Input/Output

616-72 Magnetic Tape Transport

1733-2 1 x 4 Disk Controller

856-2 Disk Drive

Program throughput can be enhanced by using the following minimum configuration:

1784-2 Computer (600-nanosecond cycle time) with 4096 words of memory

1782-2 Memory Module (600-nanosecond cycle time) with 4096 words of memory (3)

1711-4 Teletypewriter, Comment Device

1732-3 1 x 4 Magnetic Tape Controller, Input/Output Device

616-72 Magnetic Tape Drive

1733-2 1 x 4 Disk Controller

856-2 Cartridge Disk Drive

Table 1-2 is a matrix of the minimum configuration for 1700 systems.

Typical CYBER 18-20 Configuration

CYBER 18-20 Computer with 16,384 words of memory

1882-16 Memory Module with 16,384 words of memory

1811-1 Console Display

1832-4 Magnetic Tape Controller, NRZI

1860-92 Magnetic Tape Drive, nine-track

1833-1 Storage Module Drive Interface

1833-3 Storage Module Control Unit

1867-20 Storage Module Drive, 50 megabyte

TABLE 1-2. MATRIX OF MINIMUM 1700 CONFIGURATIONS

Equipment	1704	1774	1714	1784
1704 Computer (with 4096 memory words)	X			
1708 Storage Increment (three each with 4096 memory words)	X			
1774 System Controller (with 4096 memory words)		X		
1772 Memory Module (three each with 4096 memory words)		X		
1773 Direct Storage Access		X		
1775 Interrupt/Data Channel		X		
1714 Computer (with 24K memory words)			X	
1784-x Computer (with 4096 memory words)				X
1782-x Memory Module (three each with 4096 memory words)				X
1711-2 Teletypewriter	X		X	
1777 Paper Tape Station	X	X	X	
1739-1 Cartridge Disk	X	X	X	
1705 Interrupt/Data Channel	X			
1711-3 Teletypewriter		X		
1711-4 Teletypewriter				X
1733-2 Disk Controller				X
856-2 Disk Driver				X
1732-2 Magnetic Tape Controller				X
616-72 Magnetic Tape Drive				X

MAXIMUM CONFIGURATION

The following are the maximum machine configurations for MSOS.

Typical 1704 Configuration

- CPU and channels
  - -1704 Computer with 4096 words of memory
  - -1708 Memory Modules with up to seven units of 4096 words each
  - -1706 Buffered Data Channel
  - -1705 Interrupt/Data Channel
- Teletypewriters
  - -1711-2 Teletypewriters
  - -1713-2 Teletypewriter/Paper Tape Input/Output (not supported for initialization)
- Card readers
  - -1729-2 Card Reader
  - -1726/1706/405 Buffered Card Reader
  - -1726/405 Card Reader
  - -1728/430 Card Reader/Punch
- Line printers
  - -1742-1 Line Printer
  - -1740/501 Line Printer
- Disks
  - -1739-1 Cartridge Disk
  - -1733-1 Disk Drive Controller with a mix of up to eight 853 and 854 Disk Drives
  - -1738 Disk Drive Controller with a mix of up to two 853 and 854 Disk Drives
- Drums
  - -1751A to J Drum
  - -1752-1 to -4 Drum
- Magnetic tapes
  - -1731 Magnetic Tape Controller with up to eight 601 Magnetic Tape Drives (1706/10277-1 is optional)
  - -1732-1 Magnetic Tape Controller with a mix of up to eight 608 and 609 Magnetic Tape Drivers (1706 is optional)
- Communications equipment
  - -1595 Serial I/O Card
  - -364-4 Communications Multiplexer
  - -361-1 Communications Adapter
  - -361-4 Communications Adapter
  - -1745-2/1706/210 Local Terminal Controller
  - -Combination can drive a mixture of 1711-4, 1711-5, 713-10, 713-10/711-100 units.
- Paper tape readers
  - -1777/1778 Paper Tape Reader (part of unit)
  - -1721/1722 Paper Tape Reader
- Paper tape punches
  - -1777/1778 Paper Tape Punch (part of unit)
  - -1723/1724 Paper Tape Punch
- Clock timers
  - -1750/1572 Sample Rate Timer
  - -1750/1573 Line Sync Timer
  - -1750-1/1572-1 Sample Timing Unit
  - -364-4 Timer
- 1500 peripherals
  - -1544-1 to -4 Digital Input Unit
  - -1553-1 to -4 Digital Output Unit
  - -1555-1 to -3 Relay Output Unit
  - -1525-3 Analog/Digital Converter
  - -1501-10 Analog Input Multiplexer Controller
  - -1501-11 Multiplexer Channel Expansion
  - -1536-2 Relay Multiplexer Controller
  - -1502-80 Relay Multiplexer Module
  - -1590 Remote Computer Interface
  - -1547-1/2 Events Counter Unit
  - -1566 Digital-to-Analog Converter
  - -1501-81 to -83 Relay Input Channels
  - -1750-1 Computer Interface Unit
  - -1750-2 Computer Expander Unit
- Digigraphic console
  - -1744/1706 Digigraphic Controller with up to eight 274 Digigraphic Consoles

- Data set interface

  -1747/1706 Data Set Unit (interface to CYBER systems)

Typical 1714 Configuration

- CPU and channels

  -1714 Computer with 24,567 words of memory

  -1706/10277-1 Buffered Data Channel

  -10273-1 to -5 Memory Increments of 8192 words to allow expansion of memory to 65K

- Teletypewriters

  -1711-2 Teletypewriter

  -1713-1 Teletypewriter/Paper Tape Input/Output (not supported for initialization)

- Card readers

  -1729-2 Card Reader

  -1726/1706/405 Buffered Card Reader

  -1726/405 Card Reader

  -1728/430 Card Reader/Punch

- Line printers

  -1742-1 Line Printer

  -1740/501 Line Printer

- Disks

  -1739-1 Cartridge Disk

  -1733-1 Disk Drive Controller with a mix of up to eight 853 and 854 Disk Drives

  -1738 Disk Drive Controller with a mix of up to two 853 and 854 Disk Drives

- Drums

  -1751A to J Drum

  -1752-1 to -4 Drum

- Magnetic tapes

  -1731 Magnetic Tape Controller with up to eight 601 Tape Drives (1706/10277-1 is optional)

  -1732-1 Magnetic Tape Controller with a mix of up to eight 608 and 609 Tape Drives (1706/10277-1 is optional)

- Communications equipment

  -364-4 Communication Multiplexer

  -361-1 Communications Adapter

  -361-4 Communications Adapter

  -1595 Serial I/O Card

  -1745-2/1706/210 Local Terminal Controller

  -Combination can drive a mixture of 1711-4, 1711-5, 713-10, 713-10/711-100 units.

- Paper tapes

  -1777/1778 Paper Tape Station

- Clock timers

  -1750/1572 Sample Rate Timer

  -1750/1573 Line Sync Timer

  -1750-1/1572-1 Sample Timing Unit

  -364-4 Timer

- 1500 Peripherals

  -1544-1 to -4 Digital Input Unit

  -1553-1 to -4 Digital Output Unit

  -1555-1 to -3 Relay Output Unit

  -1525-3 Analog/Digital Converter

  -1501-10 Analog Input Multiplexer Controller

  -1501-11 Multiplexer Channel Expansion

  -1536-2 Relay Multiplexer Controller

  -1502-80 Relay Multiplexer Module

  -1502-81 to -83 Relay Input Channel

  -1750-1 Computer Interface Unit

  -1750-2 Computer Expander Unit

  -1590 Remote Computer Interface

  -1547-1/2 Events Counter Unit

  -1566 Digital-to-Analog Converter

- Digigraphic console

  -1744/1706 Digigraphic Controller with up to eight 274 Digigraphic Consoles

- Data set interface

  -1747/1706 Data Set Unit (interface to CYBER systems)

Typical 1774 Configuration

- CPU and channels

  -1774 System Controller with 4096 words of memory

  -1772 Memory Module with up to seven units of 4096 words each

  -1773 Direct Storage Access

  -1775 Interrupt/Data Channel

  -1706 Buffered Data Channel

  -1779 Character Handling

- Teletypewriters

  -1711-3 Teletypewriter

  -1713-3 Teletypewriter/Paper Tape Input/Output (not supported for initialization)

- Card readers

  -1729-2 Card Reader

  -1726/1706/405 Buffered Card Reader

  -1726/405 Card Reader

  -1728/430 Card Reader/Punch

- Line printers

  -1742-1 Line Printer

  -1740/501 Line Printer

- Disks

  -1739-1 Cartridge Disk

  -1733-1 Disk Drive Controller with a mix of up to eight 853 and 854 Disk Drives

  -1738 Disk Drive Controller with a mix of up to two 853 and 854 Disk Drives

- Drums

  -1751A to J Drum

  -1752-1 to -4 Drum

- Magnetic tapes

  -1731 Magnetic Tape Controller with up to eight 601 Tape Drives (1706/10277-1 is optional)

  -1732-1 Magnetic Tape Controller with a mix of up to eight 608 and 609 Tape Drives (1706 is optional)

- Communications equipment

  -364-4 Communication Multiplexer

  -361-1 Communications Adapter

  -361-4 Communications Adapter

  -1595 Serial I/O Card

  -1745-2/1706/210 Local Terminal Controller

  -Combinations can drive a mixture of 1711-4, 1711-5, 713-10, 713-10/711-100 units.

- Paper tapes

  -1777/1778 Paper Tape Station

- Clock timers

  -1750/1572 Sample Rate Timer

  -1750/1573 Line Sync Timer

  -1750-1/1572-1 Sample Timing Unit

  -364-4 Timer

- 1500 Peripherals

  -1544-1 to -4 Digital Input Unit

  -1553-1 to -4 Digital Output Unit

  -1555-1 to -3 Relay Output Unit

  -1525-3 Analog/Digital Converter

  -1501-10 Analog Input Multiplexer Controller

  -1501-11 Multiplexer Channel Expansion

  -1536-2 Relay Multiplexer Controller

  -1502-80 Relay Multiplexer Module

  -1501-81 to -83 Relay Input Channels

  -1750-1 Computer Interface Unit

  -1750-2 Computer Expander Unit

  -1590 Remote Computer Interface

  -1547-1/2 Events Counter Unit

  -1566 Digital-to-Analog Converters

- Digigraphic console

  -1744/1706 Digigraphic Controller with up to eight 274 Digigraphic Consoles

- Data set interface

  -1747/1706 Data Set Unit (interface to CYBER Systems)

Typical 1784 (CYBER 18-17) Configuration

- CPU and channels

  -1784-x Computer with 4096 words of memory

  -1782-x Memory Modules with up to 15 units of 4096 words each

  -1783-1 Expansion Enclosure (required when memory size exceeds 32 K)

  -1786-1 Memory Expansion Module (required for use of memory in 1783-1)

  -10297-1 Memory Hold Battery

  -1706/10277-1 Buffered Data Channel†

  -1785-3 A/Q Channel Adapter

  -1785-4 DSA Channel Adapter Extension

- Teletypewriters

  -1711-4 Teletypewriter, KSR-33

  -1711-5 Teletypewriter, KSR-35

  -1713-4 Teletypewriter (keyboard mode only), ASR-33

† Devices require the inclusion of the 1785-3 A/Q Channel Adapter and the 1785-4 DSA Channel Adapter.

-1713-5 Teletypewriter (keyboard mode only), ASR-35

-713-10 Conversational Display Terminal (as Teletypewriter replacement)

-711-100 Conversational Display Terminal Memory Expansion (optional)

-713-120 Slave Printer (optional usage via Conversational Display Terminal mode selection)

- Card readers/punch

  -1729-2 Card Reader†

  -1729-3 Card Reader

  -1726/1706/405 Buffered Card Reader††

  -1728/430 Card Reader/Punch†

  -1725-1 Card Punch

- Line printers

  -1742-1 Line Printer†

  -1740/501 Line Printer†

  -1742-30 Line Printer

  -1742-120 Line Printer with 595-4 Train Cartridge

- Disks

  -1739-1 Cartridge Disk††

  -1733-1 Disk Drive Controller with a mix of up to eight 853 and 854 Disk Drives††

  -1733-2 Disk Drive Controller with a mix of up to four 856-2 and 856-4 Disk Drives

  -1738† Disk Drive Controller with a mix of up to two 853 and 854 Disk Drives††

- Drums

  -1751A to J Drum††

  -1752-1 to -4 Drum††

- Magnetic tapes

  -1731 Magnetic Tape Controller† with up to eight 601 Tape Drives (1706/10277-1 is optional)††

  -1732-1† Magnetic Tape Controller† with a mix of up to eight 608 and 609 Tape Drives (1706/10277-1 is optional)††

  -1732-2 Magnetic Tape Controller with a mix of up to four 615-73 and 615-93 Tape Drives

  -1732-3 Magnetic Tape Controller with a mix of up to four 616-72, 616-92, or 616-95 Tape Drives

- Communications equipment

  -364-4 Communication Multiplexer†

  -361-1 Communications Adapter

  -361-4 Communications Adapter

  -Combination can drive a mixture of 1711-4, 1711-5, 713-10, 713-10/711-100 units.

  -1595 Serial I/O Card†

  -1745-2/1706/210 Local Terminal Controller††

  -1743-2 Asynchronous Communications Controller

- Paper tapes

  -1777/1778 Paper Tape Station†

  -1720-1 Paper Tape Reader and Punch

- Clock timers

  -1750/1572 Sample Rate Timer†

  -1750/1573 Line Sync Timer†

  -1750-1/1572-1 Sample Timing Unit†

  -364-4 Timer

  -10336-1 Real-Time Clock

- 1500 Peripherals

  -1544-1 to -4 Digital Input Unit†

  -1553-1 to -4 Digital Output Unit†

  -1555-1 to -3 Relay Output Unit†

  -1525-3 Analog/Digital Converter†

  -1501-10 Analog Input Multiplexer Controller†

  -1501-11 Multiplexer Channel Expansion†

  -1536-2 Relay Multiplexer Controller†

  -1502-80 Relay Multiplexer Module†

  -1501-81 to -83 Relay Input Channels†

  -1590 Remote Computer Interface†

  -1547-1/2 Events Counter Unit†

  -1566 Digital-to-Analog Converters†

  -1750-1 Computer Interface Unit†

  -1750-2 Computer Expander Unit†

- Digigraphic console

  -1744/1706 Digigraphic Controller†† with up to eight 274 Digigraphic Console

- Data Set interface

  -1747/1706 Data Set Unit†† (interface to CYBER Systems)

Table 1-3 is a matrix of the maximum configurations for 1700 Systems. Many various optional configurations can be constructed from the following information and matrices.

†Devices require the inclusion of the 1785-3 A/Q Channel Adapter.

††Devices require the inclusion of the 1785-3 A/Q Channel Adapter and the 1785-4 DSA Channel Adapter.

TABLE 1-3. MATRIX OF MAXIMUM 1700 CONFIGURATIONS

Equipment	1704	1714	1774	1784-x
Channels				
1783-1 Expansion Enclosure				X
1786-1 Memory Expansion				X
10297-1 Memory Hold Battery				X
1785-3 A/Q Channel Adapter				X
1785-4 DSA Channel Adapter				X
1706 Buffered Data Channel	X		X	
1706/102771 Buffered Data Channel		X		†
1705 Interrupt/Data Channel	X			
1773 Direct Storage Access			X	
1775 Interrupt/Data Channel			X	
1779 Character Handling			X	
Teletypewriters/Displays				
1711-2 Teletypewriter	X	X		
1711-3 Teletypewriter			X	
1711-4 Teletypewriter				X
1711-5 Teletypewriter				X
1713-2 Teletypewriter	X	X		
1713-3 Teletypewriter			X	
1713-4 Teletypewriter				X
1713-5 Teletypewriter				X
713-10 Conversational Display Terminal				X
711-100 CRT Memory Expansion				X
713-120 Non-Impact Printer				X
Card Readers				
1729-2	X	X	X	††
1729-3				X
1726/405	X	X	X	††
Card Readers/Punch				
1728-430	X	X	X	††
1726/1706/405	X	X	X	†
1725-1				X
Line Printers				
1742-1	X	X	X	††
1740-501	X	X	X	††

†Devices require the inclusion of the 1785-3 A/Q Channel Adapter and the 1785-4 DSA Channel Adapter.

††Devices require the inclusion of the 1785-3 A/Q Channel Adapter.

TABLE 1-3. MATRIX OF MAXIMUM 1700 CONFIGURATIONS (Contd)

Equipment	1704	1714	1774	1784-x
Line Printers (Contd)				
1742-30				X
1742-120/595-4				X
Disks				
1739-1	X	X	X	†
1733-1/853/854	X	X	X	†
1733-2/856-2/856-4				X
1738/853/854	X	X	X	†
Drum				
1751 A/J	X	X	X	†
1752-1 to 1752-4	X	X	X	†
Magnetic Tapes				
1731/601	X	X	X	†
1732-1/608/609/1706	X	X	X	†
1732-1/1706/601	X	X	X	†
1732-1/608/609	X	X	X	†
1732-2/615-73/615-93				X
Paper Tapes				
1721/1722	X			
1723/1724	X			
1777/1778	X	X	X	††
1720-1				X
Communications				
364-4	X	X	X	††
361-1	X	X	X	††
361-4	X	X	X	††
1595-10/20	X	X	X	††
1745-2/1706/210	X	X	X	†
1743-2	X	X	X	X
Clock Timers				
1750/1572	X	X	X	††
1750/1573	X	X	X	††
1750-1/1572-1	X	X	X	††
364-4	X	X	X	††
10336-1				X

†Devices require the inclusion of the 1785-3 A/Q Channel Adapter and the 1785-4 DSA Channel Adapter.

††Devices require the inclusion of the 1785-3 A/Q Channel Adapter.

TABLE 1-3. MATRIX OF MAXIMUM 1700 CONFIGURATIONS (Contd)

Equipment	1704	1714	1774	1784-x
Real-Time Peripherals				
1544-1 to 1544-4	X	X	X	††
1553-1 to 1553-4	X	X	X	††
1555-1 to 1555-3	X	X	X	††
1525-3	X	X	X	††
1501-10	X	X	X	††
1501-11	X	X	X	††
1536-2	X	X	X	†
1502-80	X	X	X	††
1501-81 to 1501-83	X	X	X	††
1750-1	X	X	X	††
1750-2	X	X	X	††
1547-1 to 1547-2	X	X	X	††
1566-22 to 1566-23	X	X	X	††
Digigraphic Console				
1774/1706/274	X	X	X	†
Data Set Interface				
1747/1706	X	X	X	†

† Devices require the inclusion of the 1785-3 A/Q Channel Adapter and the 1785-4 DSA Channel Adapter.

†† Devices require the inclusion of the 1785-3 A/Q Channel Adapter.

Typical CYBER 18-20 Configuration

- CPU
  - -CYBER 18-20 Computer with 32K bytes, MOS
  - -1882-16 MOS Main Memory Storage, 32K bytes
  - -1882-32 MOS Main Memory Storage, 64K bytes (up to 256K bytes)
  - -1874-1 ECC MOS Array
  - -1875-1 Breakpoint Controller
  - -1875-2 Breakpoint Panel
- Displays
  - -1811-1/2 Console Display
- Card readers
  - -1828-1/1829-30 Card Reader
  - -1828-1/1829-60 Card Reader
- Line printers
  - -1828-1/1827-30 Line Printer
  - -1828-1/65119-1 Line Printer
  - -1827-5/7 Matrix Printer
- Disks
  - -1833-1/1833-3 Disk Drive Controller with up to eight 1867-20 or eight 1867-10 Disk Drives
  - -1833-4 Disk Controller with up to four 1866-12 or 1866-14 Cartridge Disk Drives
- Flexible disks
  - -1833-5 Flexible Disk Drive Controller with up to two 1865 Flexible Disk Drives

- Magnetic tapes

  -1832-4 Magnetic Tape Controller with a mix of up to four 1860-72 and 1860-92 Magnetic Tape Transports

  -1860-5/6 Magnetic Tape Subsystem with up to four tape transports.

- Communications equipment

  -1843-1 Communications Line Adapter

  -1843-2 Eight Channel Communications Line Adapter

SOFTWARE

The highly modular structure of the software allows the user to customize this software and his hardware to fit his needs in an economical manner. The following discussion uses logical categories; these divisions may not necessarily be the same as those used for modular divisions. Some of the software discussed is not a part of the MSOS system, but rather works with it to give increased capabilities. These products are:

- File manager
- Report generator (RPG II)
- Sort/merge
- Macro assembler
- MS FORTRAN
- Magnetic tape utilities

Each of these subsystems has its own reference manual.

The remainder of section 1 contains a brief description of each of the major software functions which operate as part of, or under control of, MSOS:

- Monitor
- Requests (scheduling and dispatching)
- Drivers (I/O and pseudo equipments)
- Job processing
- Libraries
- Loading and linkage
- Utilities
- Maintenance
- Data Management
- System Startup
- Languages

MONITOR

The monitor is the real-time executive program for the CYBER 18/1700 MSOS. It serves as an interface between the programs and hardware. The monitor schedules use of the central processor and the I/O equipment to the various programs on a priority basis. Real-time programs that must be executed within a time limit run at high priority levels (the highest level is 15); non-real-time programs run at low priority levels (the lowest level is 0).

Core allocation is a major monitor concern. Main memory is apportioned in several ways: by priority levels, by partitions, and by protected and unprotected areas. The purpose of the first two methods is to allocate core so that important programs have a relatively large area into which they can be loaded and executed while less important programs are restricted to relatively small segments of core.

NOTE

Most foreground programs are written to be run-anywhere, so that the program need not be assigned a specific location in all systems.

The purpose of the third method is to allow a large area for background processing. This area ia allowed on a non-interference basis with the foreground programs so that if a background program is operating when a foreground program requires time, the entire background region of core is swapped onto disk. The foreground program can be executed in the space made available.

If main memory cannot be allocated because insufficient space is currently available for a program of this importance (allocation priority), the space request is queued to the core allocation routine. Allocation is retried later when some current program releases space.

The on-line programs (monitor, job processor, the library editing program, and recovery) run in protected areas of core. Batch programs (job processing) run in unprotected core. This protect feature ensures that errors in the unprotected programs do not destroy the on-line system. However, unprotected programs may make use of protected routines, such as I/O drivers, through requests to the monitor.

Unprotected programs are assisted in making monitor requests by the comprehensive program protect system. Programs running in unprotected areas are not allowed to transfer control to locations in protected areas. When unprotected programs make monitor calls, a memory protect violation interrupt occurs. The interrupt service program determines the nature of the memory protect violation and then examines the monitor request for validity. If the request is valid, the monitor proceeds with the request; if it is in error, the request is rejected.

Each interrupt line is assigned a specific priority and an associated interrupt response routine, usually a part of the I/O driver for the line's device. The priority level of the interrupt determines whether it is processed immediately or not. The typical execution priority for the system is shown in table 1-4.

If this typical priority structure is used, drivers are executed in preference to foreground programs which themselves have

TABLE 1-4. TYPICAL MSOS PRIORITIES

Priority †	Reserved For
15	Internal errors: stall alarm parity and protect violations
9-14	I/O drivers. Those drivers that are the most time-critical (e.g., card reader) have the highest priorities.
4-8	Foreground programs
2-3	Foreground idle loop (job processor, protect processor)
1 0	Background programs (completion scheduling)
-1	CPU idle loop

†See also appendix F.

preference to background programs. The interrupt handlers set an interrupt mask so that only interrupts of a higher priority can interrupt an interrupt processor. Conversely, as soon as an interrupt's processing is completed, the mask drops to a lower level. This level is that of the interrupted interrupt routine (if any) or the level below any interrupts (level 8 in the scheme shown).

NOTE

Should the running of a foreground program be critical for an application (e.g., a process control program whose failure to execute immediately could cause catastrophic harm to the user's process), such a process control program might run at a higher priority than many or all of the drivers.

These features permit the MSOS software system to execute real-time programs in response to interrupts or internal requests and to process on-line jobs on a real-time basis. Numerous real-time programs can be handled by the system with many independent concurrent processes being serviced on the basis of their priority. The computer can debug new programs in the background without danger to the process while controlling an on-line process. A manual interrupt on the computer console allows the operator to assume limited direct computer control for batch processing.

REQUEST PROCESSOR AND DISPATCHER

The request processor is a scheduler that checks each request for validity.

If the request is from a background program, an extensive validity check is given to preclude harming protected processes. A brief check is also given requests from foreground programs. If the request is valid, it is threaded to the appropriate device. In the case of a scheduler request, the scheduler either schedules the request immediately or queues the request for later processing. The former occurs only if the requesting program specifies the request to be more important than its own continued execution. In this case, the current program's operation is suspended, and that program joins the queue of other tasks awaiting execution (at, however, a higher priority than any other waiting program). In the more common case, the requested task operates at a lower priority level and takes its place in the scheduler queue according to its request priority level.

The dispatcher gains control when a task is completed or when a program is waiting for a subtask to be completed. The dispatcher executes the highest priority waiting program using these criteria:

- CPU control is passed either to the highest priority program that has been interrupted and is awaiting completion or to the highest priority scheduled request.
- If an interrupted program and a scheduled program have equal priority, the interrupted program is given CPU control.

NOTE

Numerous awaiting-execution queues exist in the systems. The most important of these are the interrupt and scheduler queues that are built up by requests and interrupt handling routines and are worked down by the dispatcher. Next in importance are the I/O driver queues discussed below. Also important are the space assignment queues discussed above.

DRIVERS (I/O AND PSEUDO STATEMENTS)

Each hardware equipment has its own driver. Some software programs (e.g., COSY or the pseudo tape unit) also have drivers. These drivers set up I/O transfers and ensure that all data transfers queued for an I/O device are processed as fast as the device can handle them.

MSOS I/O drivers are interrupt driven and DMA/DSA devices are asynchronous to mainframe operation (with the exception of the pseudo I/O drivers such as the COSY driver). Using the request parameters, the I/O drivers set up the I/O equipment for the transfer, establish buffer limits if a block transfer of data is requested, and start the data transfer. Subsequent requests to the same device are queued to this driver on a priority basis. The appropriate request module builds up the queue; the specified I/O driver works the waiting requests off the queue.

The drivers, although an integral part of MSOS, have their own separate reference manual.

JOB PROCESSOR

The job processor is a system program that monitors the unprotected core programs. This program is mass-storage resident, stored in run-anywhere form in the system library, and is read into protected core by an operator request. It allows programs to run in the background (unprotected core) when the system does not need the CPU or the background core area. The job processor runs under control of the monitor at a low priority level.

The job processor initiates and supervises all the programs running in or utilizing unprotected core, including:

Operation	Call
FORTRAN compiler, non-reentrant	*FTN
Run-time version Macro assembler	*ASSEM
COSY	*COSY
Off-line object programs	*name
Breakpoint	*B
Recovery	*SR
On-line trace	*TRACE (in program)
Relocatable binary loader	*L or *LGO
Skeleton and library builder	*SKED and *LIBILD
Library editing	*LIBEDT
I/O utilities (various)	*IOUT, *MTUT, or *DTLP
Text editor	*EDITOR
System and program maintenance routines	

The job processor can be initiated through the console manual interrupt processor.

Subsequent batch control may come from the computer console or may be assigned to standard input device. The operator may:

- Specify the input device for control statements
- Call the relocatable binary loader or load a program as an absolute file
- Instruct the loader to load a specified program
- Set the breakpoint indicator
- Set the recovery indicator
- Start execution
- Reassign standard input and output device for job processing
- Reassign standard input and output device for COSY
- Define, open, close, release, modify, and purge job processor file
- List the directory of job processor files
- Execute programs after loading or load and execute by load-and-go
- Rewind and/or unload magnetic tape units
- Temporarily suspend job processing for operator intervention
- Call the library editing program
- Call a variety of utility programs to manipulate files and transfer data from one I/O device to another

LIBRARIES

Two libraries store the programs on mass storage. The system library contains copies of all foreground programs except those residing in main memory. The main-memory-resident programs are kept in absolute format on the system image area of mass memory. From that location they are read into main memory at autoload time. The program library contains all background programs.

Foreground programs that are not main-memory-resident are written in run-anywhere form, but are stored in absolute binary format already linked to other programs. Background programs may be stored in relocatable binary format or in absolute binary format. In absolute binary format, the programs are called library files.

Several utilities allow modification of the libraries:

- A skeleton editor and library builder allow changes to the order and content of both libraries by restructuring the installation file from which the system is built.
- A library editor allows internal changes to programs on either library, including addition and deletion of programs. Without reinitialization, the system library can be expanded until all the previously provided dummy programs slots are filled with programs. The program library can be expanded as long as mass memory space is available.

A third library stores standard macro commands used by the macro assembler, which are selected during system customization. Additional macro commands may be defined by the user and stored in the library using the LIBMAC utility (see section 14). Macro commands are changed to 1700 machine code by the macro assembler.

LOADING AND LINKAGE TO OTHER PROGRAMS AND TO DATA

Programs in object form are usually written in relocatable binary format. The programs may then be stored on mass storage in several ways:

- Absolutized format for main memory programs (stored already linked in the system image region)
- Absolutized format for foreground programs (stored already linked in the system library)

- Relocatable binary format for background programs (stored in the program library and linked as they are loaded)
- Absolutized format for background programs (program files) stored in the program library. Linked programs must be absolutized and stored with the principal program.

Background programs may also be stored in relocatable binary format on cards, magnetic tape, etc. These programs are linked as they are loaded.

Foreground Program Linkage

At system initialization time, main-memory-resident programs are loaded and linked (*L, *LP, and *T statements, section 6). The programs are then saved in absolute format in the system image region of mass memory.

Also at initialization time, mass-memory-resident programs are loaded and linked (*M and *MP statements, section 6). These programs are then stored in the system library in absolute format. Entry points to these programs are listed in the CREP and CREP1 tables.

After initialization, additional foreground programs may be placed on the system library by LIBEDT (section 13) using a substitution procedure. At initialization, dummy program ordinals are usually included in the system library, giving the capability of adding programs without reinitialization until all the dummy programs are gone.

Background Program Linkage

After the initial startup, LIBEDT may be called to build the program library. Programs may be stored on the library in absolute format (program files) or in relocatable binary format (*L, *P, and *N statements, section 13). Relocatable binary programs are linked when they are loaded for execution (see *X, section 9, for linkage definition).

NOTE

A protect violation occurs when a background program references a foreground program that does not expressly allow such linkage (e.g., file manager, a foreground program, allows such linkage). The preset table in SYSDAT defines those foreground entry points that may be referenced by background programs.

If the background program is stored as an absolute file on the program library, it is linked to all its external references at the time it is initially loaded on the library (see *P statement, section 13, for linkage definition).

FORTRAN Program Linkage

If the user elects to write any type of program in FORTRAN, the subprogram linkage rules for that language must be followed. See the MS FORTRAN Reference Manual for details.

Loading

For main-memory-resident programs, the autoload procedure (see System Start-up, section 2) transfers the main memory image from mass memory to main memory and then passes control to the monitor. A foreground program is loaded at its execution address as a result of a schedule request (see SCHDLE and SYSCHD, section 3). A background program is loaded into unprotected core at its execution address as a result of a LOADER or GTFILE request.

Three types of background loading are provided:

- From the program library, a relocatable binary program is loaded, linked, and executed.
- From the program library, an absolute program (program file) is read in whole or any designated subpart, and executed.
- From the designated I/O device(s), a relocatable binary program is loaded, linked, and executed. This loading makes use of the program library directory to load routines that are linked to the requested program, but the requested program is not added to the program library by this loading operation.

Finding Data and Common Area Information

Certain routine addresses and directory addresses are kept in the communications region. These are fixed locations. Some of them can be accessed by unprotected programs.

Foreground programs can define and link to a system blank common and multiple labeled common areas. Labeled common may be preset with data (section 2).

For background programs, a separate labeled and blank common within the background area may be defined (section 2).

Requests (see sections 3, 5, and 9) make data available to/from an I/O device or from files. File manager files are available to foreground programs and with restrictions to background programs.

UTILITIES AND MAINTENANCE SOFTWARE

Three categories of programs are included in this group:

- Software programs to check hardware performance
- Software programs to aid in debugging other software
- Miscellaneous utilities

Hardware Performance Checking

The following two major programs are provided.

- The Small Computer Maintenance Monitor (SCMM) is an on-line hardware error detection program. SCMM is entered at operator option using the manual interrupt, and gives the operator the ability to test I/O equipment. The test is briefly described in section 7, and is fully described in the SCMM Reference Manual. The program cannot be used on CYBER 18-20 and CYBER 18-30 Timeshare systems.

- The engineering file is a log for saving hardware device failure information. Data is entered into the file by the appropriate I/O driver when that driver encounters an unrecoverable device error (e.g., a record cannot be written to disk). When the log is filled, the operator is notified. The contents of the log can be inspected at any time by issuing a manual interrupt followed by a request for the log contents (section 8).

Program Debugging

On-line debug (ODEBUG) may be used to debug programs at low foreground priorities. ODEBUG treats programs as files, and also includes load, dump, and alter capabilities. Since ODEBUG accesses both protected and unprotected regions of main memory, both foreground and background programs can use the debugging functions. ODEBUG is described in section 10.

The breakpoint program runs in the background and allows the operator to inspect program performance at specified decision points and to alter data and instructions at these points. The program is described in section 10.

The recovery program is provided for checking performance of batch programs after they have been executed. It is described in section 10.

Miscellaneous Utilities

The system checkout package (SYSCOP) causes the image of the failed system to be written to mass storage (bootstraps are required for this dumping operation). Then, after autoload restarts MSOS, SYSCOP systematically examines the image at low priority and prints error information. The operation of SYSCOP is described in section 11.

Library editing routines (LIBEDT) permit the user to substitute programs on both the system and program libraries, to add programs to both these libraries, and to alter absolute programs (library files) on the program library. LIBEDT is described in section 13.

Other utilities are described in section 14. These include:

- Initializer aids: SILP
- Library preparation aids: SKED, LIBILD, LIBMAC, and SETPV4
- Listing and sorting aids: LULIST, LISTR, EESORT, and OPSORT
- Program compression aids: COSY and its associated programs, CYFT and LCOSY
- I/O utilities: DTLP, DSKTAP, and IOUP
- File editing: EDITOR
- On-line program tracing: TRACE

FILE MANAGEMENT

The file manager is a complete file management system allowing the establishing and updating of files sequentially, directly, or by keywords (indices). Files are essentially free-form. The file manager consists of a request supervisor that resides in core, and a number of request processor that reside on mass storage, minimizing core requirements. Individual request processors are brought into core only as they are needed.

Core requirements for the file manager are approximately 1000 words of resident and 1200 words of allocatable core to accommodate the largest request processor. Mass storage requirements are dependent on the average record length in a file, the number of records in a file, and the length and number of key values used for indexed files.

In order to minimize mass memory I/O traffic, the file manager is designed to allow file information to remain in allocatable core until a time-out occurs, at which time the information is updated on mass storage. Users are cautioned that abnormal system stops and autoloads can destroy this information, and eventually cause fatal file errors.

If the system contains a file manager, a file validity check is performed each time the system is autoloaded. The check is preceded by the message:

CHECKING FILES -

on the system comment device. The check consists of a trace of all file space threads on mass storage. If the threads are found to be valid, an OK is printed. If errors are found, the user is given the option to continue with the autoload or to purge all the system files (i.e., all pointers to the file manager space pool are reset to a state that indicates that no files are defined). If the second option is selected, the files have to be reloaded from a user-written back-up dump.

The file manager is discussed in section 5 and is described in detail in the File Manager Reference Manual.

Job processor files are free-form background files. They are dated so that files are automatically purged on the file expiration date.

Job processor files are allocated and controlled through the file manager by use of pseudomagnetic tape logical units. Each file is essentially another tape logical unit in which data can be written, stored, and read. Use of these files provides such capabilities as:

- Establishment of dedicated files for binary and list output
- Storage of ASCII or binary information for frequently executed system routines such as LIBEDT, COSY, FORTRAN, macro assembler, etc.

The control statements for job processor files are described in section 9.

The following products and systems features use files under file manager control:

- Sort/merge facilitates copying, sorting, and merging files. Since subsorting by a series of weighted keywords is available, sort/merge provides limited text searching capability. Sort/merge uses file manager files. It is described in detail in the Sort/Merge Reference Manual.

- The report generator (RPG II) uses highly structured file manager files. RPG II produces business record-type reports; it can alter data in its structured files quickly and efficiently. This product is described in detail in the RPG II Reference Manual.

- The text editor manipulates data in one background file. Data processed by this text editor may then be stored in a job processor file for future use. Input to the file may be any free-form data whether previously declared as a file or not. The text editor is described in section 14.

SYSTEM INITIALIZATION AND STARTUP

A system initializer constructs MSOS from an installation file. At the end of initialization, the autoload sequence that places the monitor in main memory is executed. At the first startup time following initialization, the program library is customarily constructed. The initializer is described in section 6; system startup is described in section 2.

At the beginning of initialization, the operator can exercise options to write address tags or surface test the disk pack.

The main memory resident programs are read into memory, absolutized, and an image of this main memory is saved in the system image region of mass memory.

The system library is constructed as foreground programs are loaded, linked, and written onto mass memory.

After library construction is completed, initialization ends by producing the autoload (system image) area on mass storage, which makes system startup possible.

With the first startup, the program library is constructed. The system is now fully operational for foreground and background processing.

MASS STORAGE ALLOCATION

While this is the direct result of requests to the mass storage drive, certain areas are reserved on the library unit. Appendix H shows a sample layout. The File Manager Reference Manual also described the use of mass storage for file and keyword directories as well as for the files themselves.

MSOS uses word addressable storage (e.g., READ and WRITE requests) or sector addressable storage (e.g., FREAD and FWRITE requests). MSOS uses 96 word sectors. Sector address tags may be rewritten on disk as a part of system initialization.

LANGUAGES

Two principal languages are provided: 1700 assembly and MS FORTRAN.

- 1700 assembly consists of the basic 1700 machine instructions plus the enhanced machine instructions for the CYBER 18 only. The complete instruction set is found in the Macro Assembler Reference Manual. The language also includes the pseudo instructions (standard macros) for subprogram linkage, for data storage, for constant declaration, for assembler and listing control, and for definition of new macro instructions. All of these are listed in the macro library.

- MS FORTRAN. This FORTRAN is a subset of ANSI FORTRAN. There are two compilers for the 1700/CYBER 18 series. They provide the same run-time capability but differ in compilation speed and memory requirements. The MS FORTRAN is described in detail in the MS FORTRAN Reference Manual.

The monitor performs two basic functions:

- Interfaces functional programs with hardware
- Assigns system resources to tasks by priority

The following features enable the monitor to perform these functions:

- Sixteen levels of program priority – System components, including input/output equipment, are allocated on a priority basis.
- Highly interruptible structure – Interrupts are inhibited for short intervals only.
- Monitor structure – This structure allows the computer system to be time-shared by an unrestricted number of programs.
- Re-entrant structure – Monitor programs may be interrupted, called by the interrupt program, and resumed without loss of continuity.

Interrupts are handled by a common routine that saves the interrupted program's registers and priority level in a stack. For interrupts from line 0, the internal interrupt processor is entered directly. The internal interrupt processor handles abnormal conditions, such as memory parity and unprotected monitor calls. Monitor calls are passed to processing modules, which perform the required action; calls requiring input/output action are queued on a priority basis.

SCHEDULING TASKS BY PRIORITIES AND INTERRUPTS

PRIORITIES

The sixteen assigned priority levels allow the user to ensure that all critical tasks are performed in a timely manner and that the asynchronous data transfers are properly processed. To preserve interrupts (some of which must be serviced in a relatively short period of time lest a loss of data occur), the highest priorities are assigned to these interrupt handlers. To ensure that background programs do not interfere with foreground tasks, the background programs are given low priority. A typical system priority assignment is shown in table 1-4.

MSOS uses a priority scheme to determine whether task execution is to occur immediately or is to follow more important tasks. Requests for higher priority programs are executed immediately, suspending execution of the running program; requests for lower priority programs are deferred by queuing these on a first-in-first-out (FIFO) within priority level queue.

Four principal methods are used to accomplish the priority scheme:

- Scheduled programs are queued in the scheduler stack. The stack also provides the ability to defer the actual queuing of a task until a specified period has elapsed (TIMER requests).
- Interrupted programs are stacked in the interrupt stack. Since only a higher priority task can interrupt the running program, this stack is arranged on a last-in-first-out basis.
- I/O requests are queued to the I/O driver which processes them. Each of these queues is ordered on a FIFO-within-priority basis.
- Requests for space in main memory are saved in a separate queue if they cannot be accomplished immediately. These are ordered on a FIFO-within-priority basis.

Servicing stacks and queues is as follows:

1. The dispatcher works requests off the scheduler and interrupt stacks, executing the highest priority task in either stack. In the case of equal priorities in both stacks, the interrupt stack task is given control of the CPU.

   The timer routine periodically checks queued TIMER requests in the scheduler stack. If the requested delay time has elapsed, the request is queued with the other scheduler requests on an FIFO-within-priority basis.

2. The find-next-request processor (FNR) starts the highest priority task in an I/O queue as soon as the previous task is completed.

3. When space is released in main memory, the space assignment processor tries to assign space for the highest priority task in the space queue.

Requests to the monitor contain two priority values: a request priority that is the initial scheduling (threading) priority and a completion priority for continuing execution after the requested task (e.g., reading data from an I/O device) has been completed. The initial request priority determines the request's place in the appropriate queue (scheduler queue for programs, I/O device queues for I/O operations). The completion priority has a completion address associated with it. When the request is completed, that address is scheduled at the completion priority in the scheduler queue.

INTERRUPTS

A variety of devices may be attached to the 15 possible external interrupt lines available with the interrupt/data channel. Interrupts signify the occurrence of some external event and vary in importance. The computer responds to the interrupt on a priority basis. Priorities are assigned to each interrupt line at installation time.

Interrupts on any interrupt line are recognized only if the interrupt system is enabled, and the bit in the interrupt mask register (M register), which corresponds to the interrupt line, is set to one. The mask is a function of the current priority level, so that only interrupts of a higher priority level are processed.

Interrupt handling takes place in two stages: an initial noninterruptable stage during which the current state is saved, and a following stage when the interrupt processor handles the interrupt. During this second stage, the interrupt processing routine may itself be interrupted by a higher priority interrupt.

When an interrupt occurs, the interrupt handler records the current state of the A, Q, I, and P registers, overflow, and priority level in the interrupt stack. On the CYBER 18, the enhanced registers R1, R2, R3, and R4 are also saved in the extended interrupt stack.

Control is then given to the interrupt response routine. After the interrupt has been serviced. the dispatcher permits the interrupted program to be restored to execution at its original state by restoring the registers from the interrupt stack. When simultaneous interrupts occur, the hardware recognizes the lowest line number first. Mask table entries for unused interrupt lines are set to zero.

MONITOR STRUCTURE

The basic monitor, which is responsible for scheduling and for executing tasks, includes:

- Request modules to analyze requests and schedule them
- A common interrupt handler to suspend running programs and to save CPU registers so that the programs can be resumed following processing of the interrupts.
- A manual interrupt processor to handle unsolicited operator intervention
- A dispatcher to pass CPU control to the highest priority task awaiting execution
- I/O drivers to handle data transfers for peripheral devices as well as to handle special programs (e.g., pseudo tape) which are designed to appear as I/O devices
- Common data areas (SYSDAT for system parameters and I/O tables)
- Core allocation routines for volatile storage, for allocatable core, and for partitioned core

Figure 2-1 illustrates the functional relationships of the request processor, dispatcher, common interrupt handler, I/O drivers, and user programs.

The following sections describe core-resident routines in the operating system.

REQUEST ENTRY PROCESSOR

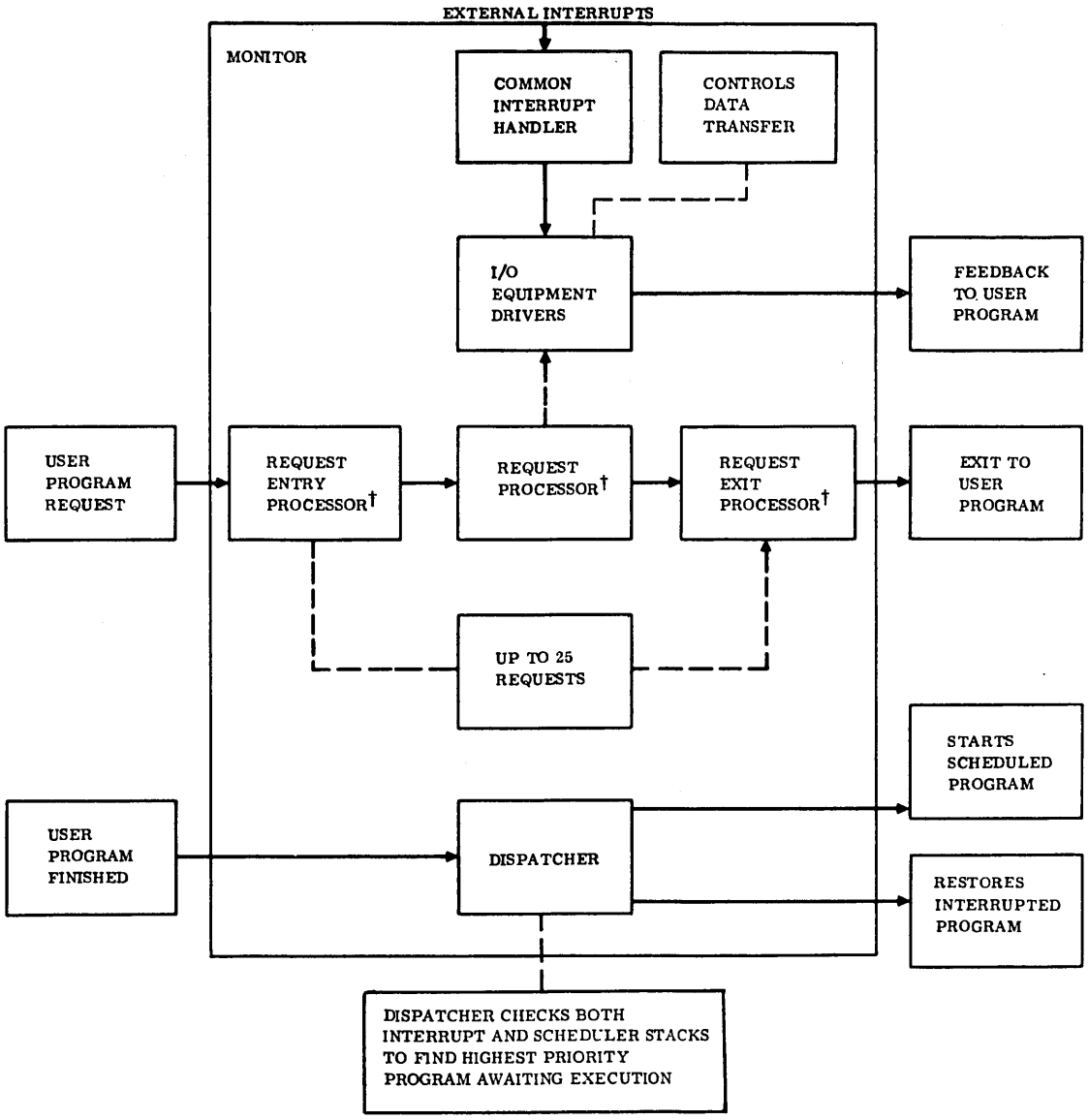
When a program makes a monitor request, the monitor stores the registers of the requesting program, examines the request for conformity with system constraints, and transfers control to the required processor. The request's parameter list defines the type of request, I/O devices required, priority, etc. Requests are described in detail in section 3.

If the program performing the request has a higher priority than the requesting program, the request is executed immediately. Otherwise, the request is queued. After a request has been queued, control returns to the requesting program if no higher priority program is waiting to run. When the current task is completed, control returns to the dispatcher (described below in this section). At this time the dispatcher inspects two stacks: the interrupt stack and the scheduler stack. The priority of the first program in each stack is checked, and the highest priority request is executed. If both stacks have an equal priority task waiting, the task on the interrupt stack is executed, since that task was already being executed before an interrupt occurred.

SCHEDULER STACK AND INTERRUPT STACK

The scheduler stack (SCHSTK) contains a waiting list of programs that have been placed there by scheduler requests. Requests are threaded together on a first-in, first-out (FIFO) basis within each priority. When a program is taken off the thread, the next program threaded becomes the top of the list. Refer to the SCHDLE and TIMER requests in section 3 for the schedule request format.

The scheduler stack also contains timer requests (requests to be delayed for a specified time). Timer requests are threaded separately from scheduler requests on four other threads. The threads contain requests whose delay time is measured in counts, tenths of seconds, seconds, and minutes, respectively. Within each thread, requests are ordered on a time-remaining-to-scheduling basis. The threads are inspected periodically by the timer request processor (TMINT). When the time remaining to scheduling elapses, this request is removed from the timer thread and placed on the scheduler thread. It is threaded using the same criterion as all other scheduler requests: FIFO by request priority.



†AFTER DRIVER STARTS PROCESSING A REQUEST, SUBSEQUENT REQUESTS ARE QUEUED. AS EACH REQUEST IS COMPLETED, THE DRIVER STARTS THE NEXT REQUEST IN QUEUE.

Figure 2-1. Monitor Block Diagram for User Programs

One more thread is contained in the stack: the thread of unused locations. When an entry is removed from scheduler thread, it is returned to the thread of empties for later use by another request. When a request is removed from a timer thread, it is threaded at the appropriate request priority in the scheduler thread.

The scheduler stack is contained in SYSDAT and is composed of four-word entries. The entry stack contains the following:

Word	15 0
0	Scheduler Call
1	Starting Address
2	Thread to Next Call
3	Q Register (Scheduler Thread) or Time-to-Go (Timer Thread)

This entry may be from a primary or secondary scheduler call as a result of a request completion or a system directory scheduler call. If the size of the scheduler list is insufficient for the system load, the location ERRCNT in the scheduler/dispatcher program (NDISP/RDISP) is nonzero. If this overflow occurs, some scheduler calls are lost.

The interrupt stack (INTSTK) consists of a waiting list of programs that have been interrupted by higher priority programs. The program status information is ordered on a last-in, first-out (LIFO) basis, since each successive interrupt is of a higher priority level than the last.

The interrupt stack is contained in SYSDAT and is made up of five-word entries, one for each priority level used in the system. The stack entry contains the following:

Word	15 14 0
0	Q Register
1	A Register
2	I Register
3	P Register
4	OV | Priority Level

Where: OV is the overflow status.

Data is related to the interrupted program.

On the CYBER 18, the four enhanced registers R1, R2, R3, and R4 are also saved, but in an extended interrupt stack. The extended interrupt stack is made up of four-word entries, one for each priority level used in the system. Each set of four-word extended interrupt stack entries is associated with the five-word interrupt stack entries at the same position in the LIFO stack.

The extended interrupt stack entry contains the following:

Word	15 0
0	R1 Enhanced Register
1	R2 Enhanced Register
2	R3 Enhanced Register
3	R4 Enhanced Register

INTERRUPT HANDLING

External interrupts transfer computer control to the common interrupt handler or to the internal interrupt handler. When an interrupt is received, the hardware transfers control to the corresponding interrupt trap, saving the program address and overflow status. An interrupt is received only if the mask register bit corresponding to that interrupt line number is set to 1.

INTERRUPT TRAP

The common interrupt handler can be entered from any of the interrupt lines. Each line has a 4-word interrupt trap. The trap region begins at location 100_{16} and ends at location $13F_{16}$.

Each trap entry is of the form:

Word	15 0
0	Entry
1	RTJ-(FE_{16})
2	pl
3	pp

or (RTJ-($F8_{16}$) for line 0)

Where: Entry is the program address applicable at the time of the interrupt.

FE_{16} contains the address of the common interrupt handler.

$F8_{16}$ contains the address of the internal interrupt processor.

pl is the priority associated with this interrupt line.

pp is the address of the primary processor that is to service the interrupt.

The overflow status is preserved in bit 15 of word 0 in 32 K mode. Each interrupt trap is assembled in the SYSDAT program and is under complete control of the user. For

example, the common interrupt handler may be bypassed simply by changing the second location of the four-word interrupt trap. Unused lines are assigned to the invalid interrupt processor, INVINT.

COMMON INTERRUPT HANDLER

The common interrupt handler (COMMON) responds to interrupts for all lines except line 0. When entered after an interrupt, the common interrupt handler resets the overflow indicator, inhibits the interrupts, and stacks (in the interrupt stack) the following information required to save the interrupted program:

- A register contents
- Q register contents
- Memory index I
- The return location and overflow indicator (if 32K)
- The priority level of the interrupt and overflow indicator (if 65K)
- The interrupt line indication (saved in memory index I after that register is saved)
- The four enhanced registers (CYBER 18 only)

The address of the next available entry in the interrupt stack is saved in location $B8_{16}$.

The new interrupt mask level is established after the required information is saved. The priority of the current interrupt is used as an index to the interrupt mask table (MASKT) and the M register is set to the value of the corresponding mask. After the mask has been loaded into the M register, the common interrupt handler enables the interrupt system and exits to the primary processor, with memory index I set to the address of the interrupt trap.

The two primary interrupt response routines provided as standard modules are the internal interrupt processor (NIPROC), for interrupts occurring only on interrupt line 0, and the external interrupt processor (LIN1V4), for interrupts occurring on line 1. All remaining line numbers use their own individual interrupt processors.

LINE 1 INTERRUPT PROCESSORS

The LIN1V4 interrupt processor (entry point LIN1V4) handles interrupts from line 1 when more than one input/output device is assigned to line 1 (normally this includes the teletypewriter or keyboard, and paper tape equipment).

Each device on line 1 is checked for a set interrupt status. If a device has interrupted, LIN1V4 enters the driver continuator for that device (the location of the driver continuator is in word 2 of the physical device table; refer to appendix C) and the driver continues or completes the input/output operation. If no device on line 1 has interrupted, a ghost interrupt is assumed and control is returned to the dispatcher.

The physical device table address for all line 1 devices must be listed in the SYSDAT table, LIN1TV4, so that devices on line 1 can be identified.

Example:

```
LIN1TV4   ADC    P1711
          ADC    P1777R
          NUM    $FFFF       END OF TABLE
```

The general interrupt processor type response described below is used in lieu of program LIN1V4, when only one device is on line 1.

GENERAL INTERRUPT PROCESSORS

Individual interrupt processing routines are used for interrupt lines that are assigned to only one device. These routines consist of setting Q to the address of the physical device table for that device and then transferring control to the driver continuator.

Example:

```
R17331    LDQ    =XP73310
          JMP*   (P73310+2)
```

The address of the interrupt response routine (R17331) is contained in word 3 of the interrupt trap for the interrupt line.

If several devices are assigned to one interrupt line, the interrupt processor must identify the device (usually by reading the status on each device) that interrupted. For some special devices the interrupt processing routine is an integral part of the driver. The address in word 3 of the trap is then set to the address of the processor for this specific interrupt.

LINE 0 INTERNAL INTERRUPT PROCESSOR

The internal interrupt processor (NIPROC) handles all internal interrupts on line 0: parity, power failure, and program protect.

Power Failure

A power failure interrupt is diagnosed by the absence of parity or program protect faults. The program saves the contents of memory locations 0 and 1 and the A, Q, and M registers. A jump to the power restoration section is placed in memory locations 0 and 1, interrupts are inhibited, and the system hangs on an $18FF_{16}$ instruction. When power is restored, the operator may continue by pressing the master clear and RUN/STEP switches (the GO button on the 1784). If auto-restart hardware is used, this sequence is done automatically. The power restoration section restores the saved memory locations and registers and exits to an optional, user-supplied power restoration routine (POWERU). If this routine is not present in the system, interrupts are inhibited and the system hangs on an $18FF_{16}$ instruction. The operator may then autoload.

Memory Parity

A memory parity interrupt is diagnosed from a test of the parity skip instruction. When this fault occurs, a diagnostic message is printed on the comment unit and an exit is made to an optional user-supplied parity fault routine (PARITY). If this routine is not present, the interrupts are inhibited and the system hangs on an $18FF_{16}$ instruction. After the parity fault problem is resolved, the system must be restarted from an autoload. If the diagnostic message PARITY DSA? appears, no parity error was encountered on the core scan. The parity fault was most likely caused by a DSA parity error.

Program Protect

A program protect interrupt is diagnosed from the program protect skip instruction. When this fault occurs, a test is made to determine if background processing is active. If no background processing is being done, interrupts are inhibited and the system hangs on an $18FF_{16}$ instruction. This condition can be caused by foreground program malfunctions that allow entry to unprotected memory. In the situation where background processing is active, the protect processor is entered (programs UPROTK, BPROTK, or in LIBEDT) to examine the system conditions and validity of the protect violation. If the violation is legal, an exit is made to the dispatcher, which allows continued execution. Illegal violations cause job termination with a diagnostic message.

DISPATCHER

When a program or program element reaches the logical end of its execution, it terminates by jumping to the dispatcher [JMP-(EA_{16}) or by use of the DISP macro]. The dispatcher determines which program is to be executed next: the top entry in the interrupt stack or the top of the scheduler list. The program with the highest priority is placed in execution or, if the two programs have equal priorities, the interrupt stack program is executed. When the program to be started is from the interrupt stack, its A, Q, and I registers and overflow condition are restored to their state at the time of the interrupt. The M register is set to the state defined by the new program's priority level.

If control is given to a program on the scheduler list, Q is set with the contents of the fourth word of the list entry [the original Q in scheduler calls, an error indication in I/O calls, or the contents of $E8_{16}$ (which is the time when the schedule request is made) for timer calls] and the mask or M register is set to correspond to the new priority (word 1 of the scheduler list entry). The other registers and overflow are arbitrary. The interrupt system is enabled before the program is placed in execution.

The dispatcher function is performed in the NDISP or RDISP module of the monitor, which also contains the scheduler. NDISP is the normal scheduler/dispatcher; RDISP is the re-entrant FORTRAN scheduler/dispatcher. The two variants perform identical functions, except that RDISP also has the capability to handle FORTRAN multiprogramming. If RDISP is used, the SYSDAT program must contain FMASK, which indicates allowable FORTRAN levels (bit 0 = level 0, etc.), and the list (FLIST) of entries in the re-entrant ENCODE/DECODE runtime that require preservation in volatile storage to achieve runtime re-entrancy. Each time a FORTRAN level is interrupted, the memory locations whose addresses are in FLIST are saved in volatile storage. When a FORTRAN level is re-entered from a higher level, the memory locations are restored.

MANUAL INTERRUPT PROCESSOR

The manual interrupt processor (MINT) responds to interrupts generated by the use of the manual interrupt button. The program prints the message, MI, on the system's comment output device and requests input of the desired operation from the comment input device.

CAUTION

The response to the MI typeout is required to terminate a level 3 loop in MINT. System programs running at level 3 are suspended until the reply to the MI typeout is entered.

The entry is of two basic forms: preceded or not preceded by an asterisk. Entries preceded by an asterisk are only allowable during job processing (section 9), with three exceptions:

*BATCH	Initiates job processing
*	Treated as a do-nothing command in the absence of the job processor
*R, LU	Restores a logical unit that has failed (section 9)

Illegal entries cause the error message JP05 to be displayed.

Entries not preceded by an asterisk cause the scheduling of the manual input processor program (MIPRO), which resides in the system library. The Q register contains the location of the input data character string.

Manual interrupts entered while another MI request is being processed are ignored.

MANUAL INPUT PROGRAM

The manual input program (MIPRO) is a mass-storage-resident system library program. MIPRO handles all entries following the MI typeout that do not begin with an asterisk.

There is a set of standard entry mnemonics that are diagnosed by the MIPRO program. The following list defines the entry sequence.

Input	Description
DACS	Enters the initialization package for the AUTRAN 2 product. This mnemonic should be used only if AUTRAN 2 is present in the system. Consult the AUTRAN 2 Reference Manual for usage instructions.

Input	Description
DATE	Allows the entry of a new date/time value via the system library program TDFUNC. The request is ENTER DATE/TIME MMDDYYHHMM Where: MM is month DD is day YY is year HH is hour MM is minute An entry, e.g.,: 0619761126 is acknowledged by: DATE: 19 JUN 76 TIME: 1126:00
DB	Initiates the on-line debug package. Refer to section 10.
DX	Terminates the on-line debug package I/O operations. Refer to section 10.
EF	Prints the system engineering file information. Refer to section 8.
EFLU	Prints the system engineering file information for one logical unit. Refer to section 8.
EFMM	Prints the core-resident mass memory error information for the engineering file. Refer to section 8.
SCMM	Initiates the on-line Small Computer Maintenance Monitor. Refer to the SCMM Reference Manual.
SYSCOP	Initiates the system checkout package. Refer to section 11.
TIME	Prints the current system date and time: DATE: 11 JAN 76 TIME: 1146:26
TOFF	Stops the system hardware timer, if present in the system.
TON	Starts the system hardware timer, if present in the system.
TSUT	Enters the utility package for the TIMESHARE 3 product. This mnemonic should be used only if TIMESHARE 3 is present in the system. Consult the TIMESHARE 3 Reference Manual for usage instructions.
VERIFY	Initiates the system verification test package, which is supplied with the released system for basic system validation.
WROF,lu	Selects write ring off for specified magnetic tape simulator lu
WRON,lu	Selects write ring on for specified magnetic tape simulator lu
CLEAR	Clear all file manager space
EDTLP	Extedned DTLP - Save/Restore mass memory
FMAP	File Manager File Mapping Program for TIMESHARE 3.0.
MIPRO	Add description of new functions.
SYEXER	TIMESHARE 3.0 system exerciser
TSTATS	TIMESHARE 3.0 status program
?	Entries preceded by a question mark cause entry to the 1700 IMPORT product. This mnemonic should be used only if IMPORT is included in the system. Consult the 1700 IMPORT Reference Manual for usage instructions.
=Sxxx,y,zzzz	Allows the user to schedule mass-resident system library ordinal programs. Where: xxx is the decimal ordinal number. y is the hexadecimal execution priority (4 bits). zzzz is the optional hexadecimal Q-register parameter that is passed to the ordinal program. NOTE: All leading zeros are required. The zzzz parameter and preceding comma are optional.

Illegal =S parameters, unlinked command processors, or commands entered outside the above mnemonic list cause an error diagnostic. Rejects encountered in the hardware timer commands cause an error diagnostic.

INPUT/OUTPUT DRIVERS

Each device in the system is associated with a device driver, which is the only piece of software that is allowed to give direct commands to the device. The driver controls execution of the read, write, and motion requests that are passed to the monitor by the user programs.

Each driver normally has three entries: initiator, continuator, and time-out (error). Variable parameters relating to the device and the driver's working storage are contained in the physical device table, in a format common to all drivers (refer to appendix C). Functionally, the initiator initializes the working storage in the physical device table and initiates input/output on an idle device; the continuator drives the device to perform the actual requested task. If the diagnostic timer detects a device hang-up (the expected interrupt does not occur within the specified time), a timer entry is entered.

Whenever a program requires input or output (I/O) for data it is processing, it makes a monitor request to effect the desired transfer. The monitor queues the request for processing by an I/O driver. A driver may handle more than one device of the same type, but requires a separate physical device table for each device.

When a request is queued, the appropriate request processor determines if the driver is available. If the driver is not busy, its initiator is scheduled. The request exit processor returns control to the caller.

Upon entry to the initiator, a call is made to the find-next-request routine, which decodes the requestor's parameter list and places the information in the physical device table. The driver initiates the I/O operation and selects some interrupt condition (EOP, data, etc.). A diagnostic clock value also is set in the physical device table. This value sets the maximum amount of time allowed to complete the operation. Then an exit is made to the dispatcher.

When the I/O device completes the operation, an interrupt is generated. When the interrupt mask is set to a priority lower than the driver's interrupt priority, an interrupt occurs that stops the program currently being executed. The common interrupt handler saves the program's registers and overflow state in the interrupt stack, and passes control to the interrupt response routine, which enters the driver's continuator entry point. The driver acknowledges the interrupt and performs the I/O command or, if the request is complete, the complete request routine is called, followed by a jump to the initiator.

If there is a hardware malfunction and the device fails to give an interrupt at the end of an operation, the time-out entry is scheduled by the diagnostic timer routine when the clock value in the physical device table has expired (if a timer is present in the system). The driver uses the MAKQ routine to set the error flags and calls the device error logging routine. If the logical unit number is not that of a diagnostic logical unit, the alternate device handler may be called by a jump or by a scheduler request. If the request was performed on a diagnostic logical unit, the complete request routine is called instead of the alternate device handler, followed by a jump to the initiator part of the driver. The diagnostic clock is set negative when a device is inactive.

FIND-NEXT-REQUEST (FNR)

The find-next-request (FNR) subroutine is used by all driver initiator modules to find the next request for a device and to fill the physical device table with information from the request. FNR is entered by an indirect return jump through $B5_{16}$ with the core address of the physical device table entry in the I register.

Device Shared

FNR scans the logical unit table, starting with logical unit 1, to locate other logical units related to the same device. When a logical unit with a waiting request is encountered, FNR initiates the input/output device in the same manner as unshared devices. The lowest numbered logical unit with a request waiting for that device has its request processed, even if a higher numbered logical unit has a higher priority request. If no requests are waiting on a device, FNR exits to the caller at the address of the call plus one.

Device Not Shared

FNR examines the queue to obtain the next request. If none exist, FNR exits to the caller at the address of the call plus one. If another request is found, FNR updates the queue, fills the physical device table, and returns to the called at the address of the call plus two. Upon return, the I register is unchanged and the A, Q, and overflow registers are destroyed.

COMPLETE REQUEST (COMPRQ)

The complete request (COMPRQ) subroutine is entered by an indirect return jump through B6 from input/output drivers to complete requests. This causes interrupts to be inhibited and the completion address to be scheduled with the error field from the physical device table, replacing the error indicator (v field) of the I/O request parameter list for logical units that do not share devices. Q is set negative if an error occurs. The request parameter list (which contains a request code designating it as an I/O call) is interpreted as a secondary scheduler call by setting bit 15 of the first word to 1. The scheduler resets it to 0 and the device is released from its request assignment. When the driver has completed the request, control is given to the dispatcher. The dispatcher then passes control to the highest priority interrupted program or scheduled program. The latter might be the completion address if one was specified and is the highest priority program awaiting execution.

The complete request is entered by a return jump to COMPRQ, which terminates the request by executing the following:

1. Resets the diagnostic clock counter (EDCLK) to $FFFF_{16}$
2. Transfers the error field in the physical device table (ESTAT1) to the v field of the request
3. Clears the operation-in-progress bit (EREQST)
4. Clears the thread and returns to the driver if there is no completion address (C = 0)
5. If there is a completion address, schedules the completion address, passing any error condition in Q and in the v field of the request, and returns to the driver.

ERROR FLAG SET-UP (MAKQ)

The MAKQ subroutine is used by the drivers to set up the v field of the logical unit word of the request and to place the address of the word following the last valid data into the last word of the caller's buffer.

COMPLETION ROUTINES

The completion address specified in the parameter list is scheduled when the I/O operation has been completed. Upon entry to the completion routine, the Q register contains the error status, if any, of the I/O operation, and the A register contains the address of the parameter list. If an error has occurred, the Q register is negative (bit 15 = 1) and should be tested by the completion routine. Thus, the original state of the registers at the time of the monitor request is not preserved. The priority level is that specified by the completion priority in the monitor request parameter list.

Completion routines that are in unprotected core are always executed at priority level 1. Upon completion of an I/O request, the request parameter specifying number of words is returned to the user's request from the request stack. This allows changes to this parameter by certain drivers (OCR devices).

INPUT/OUTPUT HANG-UP ERRORS

An input/output hang-up error occurs when a driver fails to get a completion interrupt on an operation that it initiated. The diagnostic timer module detects hang-ups. The following features must be available for proper operation of the diagnostic timer module or these errors cannot be detected.

- A hardware device that gives periodic interrupts to measure time or a software pseudo-timer module
- The MSOS timer request module (TMINT)
- The MSOS diagnostic timer module (DTIMER)

The driver establishes a time differential (in increments of seconds) for each input/output operation; when this period elapses, a hang-up is declared. This differential is entered in the physical device table slot for the device. Each time the diagnostic timer module is executed, it decrements the time differential. When the differential becomes negative after decrementing, a hang-up is declared. If the time differential is negative before decrementing, either the operation is complete or no operation has occurred.

When a hang-up occurs, the diagnostic timer accesses the physical device table entry for that device to obtain the driver core location to be executed in case of a hang-up. This location is executed by a SCHDLE request at the same priority level as the driver. Q contains the core address of the physical device table entry for the device. The driver takes any necessary action to clear the device involved in the hang-up. If recovery is not possible, the error is logged in the engineering file and control is transferred to the alternate device handler. The logical unit number and error code parameters are passed to the alternate device.

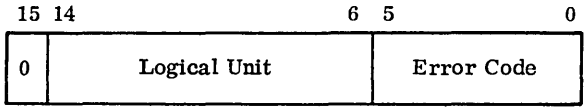
Initially, the diagnostic timer is executed after system start-up. Thereafter, it is periodically reactivated by a TIMER request. The frequency of execution is dependent on a parameter internal to the diagnostic timer program (normally one second).

The devices to be checked by the diagnostic timer are specified by a table of physical device table addresses. This table (DGNTAB) is included in SYSDAT.

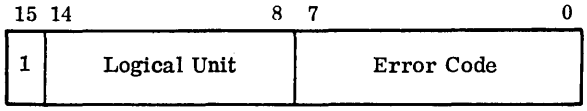
ALTERNATE DEVICES

When a driver detects an irrecoverable failure of a data transfer or a motion request, the following actions take place:

1. The driver sets the error field in bits 15 through 13 at word 9 (ESTAT1) of the physical device table for the device.
2. The controller hardware is cleared and an error word is set in the Q register. Two formats are used. If bit 15 is 0, Q is interpreted as:



If bit 15 is a 1, Q is interpreted as:



3. The driver transfers control to the alternate device handler by a jump or a scheduler request, with the error word in the Q register.

Typical errors (see appendix J for error codes) are as follows:

- Input/output hang-up (diagnostic timer)
- Alarm
- Parity error
- Checksum error
- Internal reject
- External reject

The alternate device handler determines if there is an operational alternate physical device; if so, the request for the device that failed is assigned to it at the priority level of the driver operating the device that failed. The logical unit that failed is marked down and the alternate logical unit is set active (bit 13 of LOG1 is set). The request is then rethreaded to the requested logical unit. If no alternate exists, the program reschedules itself at a low priority level to request operator intervention. In either case, all message output is executed from a low priority level section of the program.

The alternate device handler continues to assign alternates to devices that failed without waiting for completion of input/output messages. Therefore, the buffer table

(ALTERR) must be provided to store the error words on entry. This table is included in SYSDAT. The size of the table is included in the first location of the table and is equal to the number of devices that can malfunction at one time. If this table size is not adequate for the system, the alternate device program causes the system to hang on an ($18FF_{16}$) instruction (entry SYFAIL) in SYSDAT when the table is filled. Identical device failures are not accumulated in the error table.

If the alternate is also inoperative and it does not specify another alternate device, the procedure is the same as if no alternate were available for the original unit. Alternate devices for this device are checked until the handler determines that none is available. Note that non-physical devices serviced by drivers (e.g., pseudo-tape) cannot have an alternate device.

If an operative alternate is found, pointers are set so that requests from the failed device are automatically transferred to the alternate. The comment device displays the following message:

L,lu FAILED ec

ALT, aa

Where: lu is the logical unit number of the failed device.

ec is the error code.

aa is the logical unit number of the alternate.

If no alternate is found, the handler issues the following diagnostic:

L,lu FAILED ec

ACTION

Where: lu is the logical unit number of the failed device.

ec is the error code.

The operator must respond to the error with one of the following and press RETURN.

RP Repeat processing – directs the request to repeat

CU Continued with device up – reports the error to the requesting program; the device is allowed to continue processing requests.

CD Continue with device down – causes any future programs calling the device to be informed of the failure upon completion; the error is reported to the calling program and the device is marked down. No subsequent attempt is made to operate this device. The message LU xx DOWN is printed on the comment device.

DU Discontinue, up – activates CU and terminates the current job being processed; the input unit attempts to slew to the next job to be executed.

DD Discontinue, down – activates CD and terminates the current job being processed; the input unit attempts to slew to the next job to be executed.

If job processing is not in progress when the DU and DD options are selected, no action is taken, the word ACTION is retyped, and another option may be selected.

Mass storage device drivers do not use the alternate device handler. Mass storage device errors are logged in the engineering file (section 8).

The comment device must never be marked down because it is required to bring devices back up once they are operational. The dummy device driver, acting as an alternate for the comment device, restores the downed comment device.

If a downed device is requested by a program and if this device contains no alternate, the following message is typed on the comment device:

L,lu DOWN

Where: lu is the logical unit number

This message occurs only the first time it is requested after being downed.

The completion address is always scheduled with an error. The requesting program should not repeatedly request downed units.

DUMMY DRIVER (DUMMY)

When the dummy driver is defined as an alternate, it completes failed I/O requests which have an error indication without operator intervention. The dummy driver also restores the failed device to an up condition. This technique is always used to prevent system hang-ups on the standard comment device. Programs should always check for I/O errors at their completion address and take appropriate action when errors occur (bit 15 of Q = 1).

The dummy driver logical unit can also be used in place of any other system logical unit. In this case, the I/O request is completed with no error. This facility is useful for slewing through records.

MASS-STORAGE-RESIDENT DRIVERS

Standard CYBER 18/1700 drivers released with MSOS have the capability to operate from memory or mass memory residency, with the exception of the following drivers which may operate only from memory:

- 1711/1712/713 Teletypewriter, CRT, and 1811 Console Display
- 1713 Teletypewriter Keyboard
- 364-4 Communications Multiplexer
- Mass memory device
- 1747 Data Set Controller
- 1744 Digigraphic Controller

- 1500 Series
- Software buffer
- Dummy

All mass-storage-resident drivers execute in a shared fixed buffer area located in SYSDAT. The allocation of this buffer is controller by a core-resident executive routine, MMEXEC. It is possible to load either one or two drivers in the buffer, depending on its size.

The core buffer should be at least as large as the largest driver that is used in the system. The maximum size required, which would allow two drivers in core simultaneously, is the combined size of the two largest drivers in the system. Whenever the DCOSY driver is used in a system, the maximum size criterion (the two largest drivers) should be used since DCOSY makes I/O requests upon another driver, which has to be in core at the same time to complete the request for the COSY driver.

When a driver is mass-memory-resident, the driver's physical device tables must declare their initiator, continuator, and time-out entry addresses to be the corresponding entries in MMEXEC; i.e., MASDRV, MASCON, and MASERR.

When an I/O request is made upon a mass-memory-resident driver, control is routed to the entry point MASDRV. MASDRV determines if the driver is already in core and passes control to it. If the driver is not in core, it is determined if there is sufficient core available in the buffer for the new driver. If so, the driver is read in from mass memory and placed in execution. When there is not sufficient core for the driver, it is queued for later execution when space is released by drivers that currently occupy the buffer.

When a driver completes its I/O for all of the devices it controls, it releases its space in the core buffer by jumping to MASEXT. At this time MMEXEC resets all initiator, continuator, and time-out addresses for this driver's physical device tables to point to the corresponding entries in MMEXEC. If any other drivers are waiting in the queue, the first one encountered is read from mass memory and placed in execution.

When MMEXEC enters a driver, the Q register contains the physical device table address and the A register contains the first word address of the driver.

The mass memory location and size of each mass-memory driver must be contained in words 13 and 14 of the physical device table. These parameters are supplied by *S initializer control statements during system installation. If a driver is core-resident, its mass memory size must be set to zero and its length to $7FFFF_{16}$. (This is not required for mass memory device drivers such as disk or drum.)

TIMER REQUEST PROCESSOR AND SYSTEM TIMEKEEPING ROUTINES

The optional timer package, which is located within the monitor, functions in conjunction with a system hardware timer (for accurate time delays) or a software pseudo timer (for approximating time delays). If one of these time bases is present in the system, the system can execute timer requests (TMINT), compute time of day (TOD), and provide auxiliary time/date calendar functions (TDFUNC).

SYSTEM TIMERS

MSOS supports a variety of system timers that allow the timer monitor requests to be implemented. The system timer is started (if present) by the SPACE program routine RESTRT when the system is autoloaded. Subsequently, the timer may be turned off or restarted by entering the commands TOFF and TON, respectively, through the manual interrupt processor. The timer type is defined in SYSDAT by the following codes:

0 No timer present
1 1573 Line Synchronized Timer
2 1572 Sample Rate Generator
3 1572-1 Line Synchronized Timer
4 1572-1 Sample Rate Generator
5 364-4 Communications Multiplexer Timer
6 Pseudo software timer
7 10336-1 Real-Time Clock
8 CYBER 18 Real-Time Clock

Timer types 1 and 3 use the power source line frequency as a time base. Types 2, 4, 5, 7, and 8 use a free-running oscillator. Timer type 6 is noninterrupt driven. The time base of type 6 is the rate at which the memory cell can be counted down in the level 1 and 2 idle loops. This time base is a function of system loading and, therefore, should not be relied upon as an accurate time base.

When the 364-4 is used as the basic system timing device, the multiplexer must be adjusted to provide 60 interrupts per second, which causes the timer interrupt response to be executed at every interrupt of the multiplexer. This response routine is designed so that the 364-4 Driver is entered on every other interrupt, which corresponds to 30 character-per-second operation. The timer response processor is executed 60 times per second.

Users are cautioned that interrupts from timer types 2, 4, 5, 7, and 8 are generated from free-running oscillators. Consequently, the time of day is not synchronized to wall-clock time, which is based on line frequency.

TIMER REQUEST PROCESSOR

The timer request processor (TMINT) is used to process scheduler requests that are to be executed following a time delay. Delays may be specified in increments of counts, tenths of seconds, seconds, and minutes. Four threads are maintained for the requests, one for each type of increment. The top of each of these threads is contained within the TMINT program, with the threaded requests located in the scheduler list. The timing delay is not designed to be precise, but to provide for a delay of at least the specified number of increments. In addition to the specified delay, the portion of the increment remaining to be counted down is also added to the delay.

Examples:

1. A program is scheduled for a delay of 10 seconds.

2. The seconds counter within the TMINT program is 40 counts into the next second (60 counts per second).

3. The increment quantity word of the request is decremented each time the seconds counter expires. The first decrement operation occurs in 20 counts.

4. The increment quantity word is decremented until it is negative. In this example the request is moved to the scheduler list thread after 10 full seconds plus the 20 counts remaining in the seconds increment portion at the start of the timer request.

5. In this example a request for a delay of 0 second would be executed 20 counts later.

To control the system scheduler overhead, the number of timer requests that can expire on the same count interrupt is a SYSDAT parameter (NSR). If more delays expire than the quantity allowed, the request is rethreaded to the counts thread for servicing on the next count interrupt.

TIME-OF-DAY PROGRAM

The system time-of-day program (TOD) is initiated during system start-up (SPACE) and keeps system wall-clock time, based on the time value entered during the system start-up sequence. The TOD program operates by making 30-count timer requests to update the time. The time parameters are kept in the SYSDAT program. A user can cause an immediate time update by making the subroutine call:

RTJ+ TOD or

CALL TOD

The routine is re-entrant and preserves all the caller's registers.

The following entry point time information is available:

HORTO	Hour (integer)
MINTO	Minute (integer)
SECON	Second (integer)
CONTA	Count (integer)
HORMIN	Military time (integer)
TOTMIN	Total day minutes (integer)

At the beginning of each new day, the time/date function program is scheduled to update the system date.

TIME/DATE FUNCTION PROGRAM

The time/date function program (TDFUNC) is used to set and print the system date and time. The program can be used through MIPRO to enter a date/time value or to print the current date/time. The date is also automatically updated at the start of each new day in conjunction with the TOD program. The date parameters are kept in the SYSDAT program.

The following is the entry point date information available:

YERTO	Year (integer)
MONTO	Month (integer)
DAYTO	Day (integer)
AYERTO	Year (ASCII)
AMONTO	Month (ASCII)
ADAYTO	Day (ASCII)

SYSTEM START-UP

The system is started from an inactive condition by a master-clear-autoload-run operation. This causes the autoload program, which resides on the mass memory library unit, to be loaded into core and executed. The function of the autoload program is to read the core-resident system image from the library unit to core and then to transfer control to the restart routine.

The restart routine is contained within the SPACE program and resides immediately adjacent to the space request processor. This has the effect of allowing the restart routine to execute in allocatable core as a core-resident routine, but does not require core once the system is in operation.

The restart routine performs the following tasks:

1. Sets up the allocatable core area table, located in SYSDAT

2. Protects and unprotects all appropriate core locations

3. Sets up the initial overlay program length for LIBEDT and the protect processor in their respective system directories

4. Starts the system hardware timer and schedules the diagnostic timer and time-of-day programs

5. Prints a message on the comment device that contains the current system PSR level and the date that the system was built

6. Prints a message on the comment device to request that the program protect switch be enabled

7. Prints a message on the comment device that contains the system identification

8. Prints a message on the comment device that indicates the addressing mode of the machine (e.g., 32K or 65K)

9. If the system contains a file manager, a file validity check is performed. If the files are not valid, the user has the option of continuing or purging them

10. Initiates the system program TDFUNC, which requests an entry of the current date and time

11. Transfers control to the system idle loop

The memory occupied by restart reverts to allocatable core at the conclusion of the routine.

The following is a typical system comment device output that follows an autoload:

Message Input	Comments
MSOS 5.0 – PSR level 110 01/23/76	PSR levels define the software update configuraiton. A new level is published monthly.
SET PROGRAM PROTECT SYSTEM IDENTIFICATION 32K MODE	Operating mode
CHECKING FILES – ERRORS	File manager is in the system.
CLEAR ALL FILES? (YES/NO) YES	The operator must choose to accept bad files or purge them. The operator accepts files.
ENTER DATE/TIME MMDDYYHHMM	
0209260905	The operator sets the new time base for CPU
DATE: 09 FEB 76 TIME: 0905:00	The CPU replies with plain text time

I/O CHANNEL ALLOCATION

Two I/O channels may be allocated:

- A/Q channel. A register is used to transfer data, Q register indicates I/O channel used.
- 1706 Buffered Data Channel on 1700 Systems

A/Q CHANNEL ALLOCATION

The CYBER 18/1700 Computer I/O is an unbuffered operation when performed via the A/Q channel. For devices where the data is contained on transportable media, such as cards, paper tape, and magnetic tape, and where the controller does not buffer the data, data can be lost if inadequate response time is provided to service a data interrupt. To avoid lost data, the drivers of these devices must run at a higher priority than the system hardware timer, if present.

The following devices are subject to this data handling restriction on the 1700 systems:

- 1721/1722/1777 Paper Tape Reader
- 1723/1724/1777 Paper Tape Punch
- 1728/430 Card Reader/Punch
- 1729-2 Card Reader
- 1729-3 Card Reader
- 1731/601 Magnetic Tape
- 1732-1/608/609 Magnetic Tape
- 1713 Teletypewriter Paper Tape I/O
- 1711/1713/713 Teletypewriter/CDT

The following devices are subject to this data handling restriction on the CYBER 18 systems:

- 1827-30/65119-1 Line Printer
- 1829-30/60 Card Reader
- 1811-1 Console Display

The A/Q channel allocator (ALAQ) program is provided to allocate the A/Q channel if more than one of these devices is present in a system. The A/Q allocator is functionally equivalent to the 1706 allocator except that the ALAQ request entry is RQAQ instead of RLAQ.

1706 BUFFERED DATA CHANNEL ALLOCATION

The 1700 computer system uses the 1706 Buffered Data Channel to buffer data transfers involving A/Q devices. This is useful for decreasing software overhead on input/output operations. Up to three 1706 units may be present in a system, with up to eight devices on each unit. Since a 1706 may only transfer data to one device at a time, the buffered data channel allocator program (AL1706) is used to queue the use of the 1706 by drivers whose devices share one 1706. The following drivers are designed to accommodate a shared 1706:

- 1726/1706/405 Card Reader
- 1731/1706/601 Magnetic Tape
- 1732-1/1706/608/609 Magnetic Tape

The following drivers control devices which each require a dedicated 1706 and cannot use the 1706 allocator:

- 1747/1706 Data Set Controller
- 1744/1706/274 Digigraphic Console

The 1706 allocator must be requested prior to a 1706 operation. The request is:

```
RTJ+ RQ1706
```

CORE MANAGERS

VOLATILE STORAGE ASSIGNMENT

Volatile storage (VOLBLK) is the storage area located in SYSDAT that is reserved for the allocation of small blocks of data storage for re-entrant routines (may operate at more than one level at the same time).

Volatile storage is available only to protected programs. At least three locations must be requested and all system interrupts disabled prior to entry at VOLA and VOLR.

The volatile storage area acquired must be released at the same priority level at which it was acquired. The requesting program and its possible accompanying program sequence must not go to the dispatcher prior to the release of the volatile storage area.

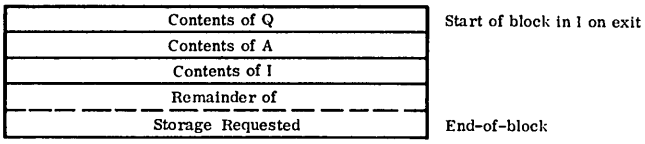
A request for more volatile storage than is available constitutes a catastrophic condition. The volatile storage assignment program enters OVFVOL with the following in the A and Q registers:

A	Amount of overflow in words
Q	Base address of the interrupt stack

OVFVOL clears the M register and writes OV on the comment device. No further action can be taken and the system hangs ($18FF_{16}$ instruction). The OV error is caused by incorrect set-up or use of the system.

The size of VOLBLK is the SYSDAT parameter. A block of storage is assigned with the entry point VOLA and released with the entry point VOLR. Both entry points are entered by an RTJ with interrupts inhibited.

On entry to VOLA, the block size (minimum of three words) is contained in the word following the RTJ. VOLA assigns specified locations and fills the first three locations of the block with the contents of Q, A, and I as follows:



On exit from VOLA, the I register contains the address of the start of the assigned block.

Example:

A subroutine is entered with 1 in A, 2 in Q, and 3 in I. Eight words of volatile storage are requested as intermediate storage.

```
        Subroutine              Comments
ENTRY   NUM  0                  Subroutine entry
        EQU  VOLA ($BB)
        EQU  VOLR ($BA)
        IIN  0                  Inhibit interrupts
        RTJ- (VOLA)             Volatile calling
        NUM  8                  sequence
        LDA* ENTRY              Get return address
        EIN  0
        STA- 3,I                Save entry in volatile
          .                     Process call
          .
          .
        IIN  0
        LDA- 3,I
        STA* ENTRY              Restore return address
        RTJ- (VOLR)
        EIN  0
        JMP* (ENTRY)            Return, registers
                                restored
```

This example could also have been coded using library macros for volatile storage, as in the following:

```
        Subroutine              Comments
ENTRY   NUM  0                  Subroutine entry
        VOLA 5,ENTRY            Get eight words (macro
                                adds three words to
                                number requested)
          .                     Process call
          .
          .
        VOLR ENTRY
```

An optional form for the VOLR macro is

```
VOLR ENTRY,x
```

Where: x is an increment to be added to the return address.

On return from VOLA, a block of eight volatile storage locations has been assigned and words 0 through 2 have been filled. The program stores word 3 and later uses the remaining words.

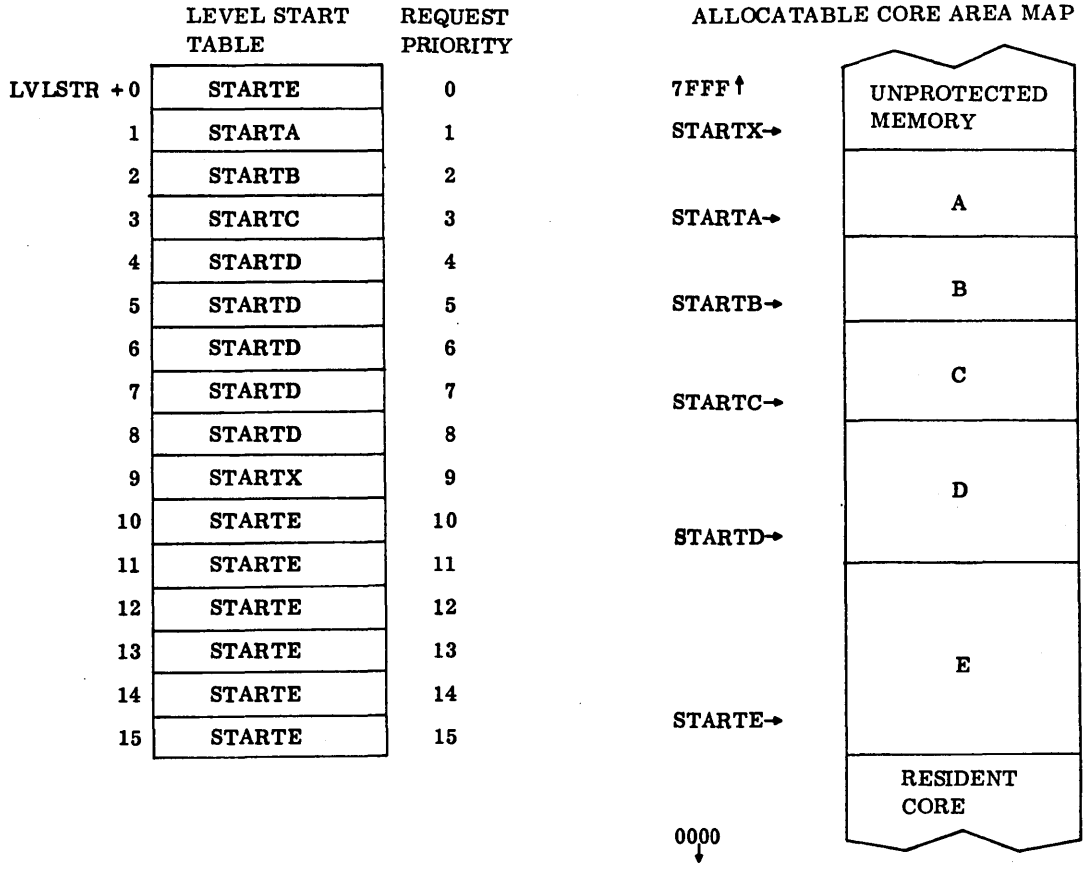
Location	15 … 4	3	2	1	0
LOC+0	Original contents of (Q)	0	0	1	0
1	Original contents of (A)	0	0	0	1
2	Original contents of (I)	0	0	1	1
3	Return Address (Saved by Requesting Program)				
4 … 7	Temporary Storage				

The I register contains the core location represented by LOC. The contents of A and Q are the same as an entry to VOLA. On entry to VOLR, I must contain LOC. On return from VOLR, the eight locations of volatile storage have been released. The contents of the A, Q, and I registers are replaced with the contents of the first three locations of the released block.

ALLOCATABLE CORE

The SPACE request (section 3) allocates space for the operation of mass-storage-resident programs in protected core. The core allocator used in making a SPACE request is not available to unprotected programs.

The core area from which space is assigned is defined within the program SPACE. The area in the block is divided to correspond to the level table, LVLSTR, as shown in figure 2-2.



NOTES:
1. PROGRAMS OF REQUEST PRIORITY LEVEL 1 CAN EXECUTE ONLY IN AREA A.
2. PROGRAMS OF REQUEST PRIORITY LEVEL 2 CAN EXECUTE IN AREA A AND/OR AREA B.
3. PROGRAMS OF REQUEST PRIORITY LEVEL 3 CAN EXECUTE ANYWHERE IN AREAS A, B, OR C.
4. PROGRAMS OF REQUEST PRIORITY LEVELS 4 THROUGH 8 CAN EXECUTE ANYWHERE IN AREAS A, B, C, OR D.
5. PROGRAMS OF REQUEST PRIORITY LEVEL 0 OR LEVELS 9 THROUGH 15 CAN EXECUTE ANYWHERE IN THE ENTIRE ALLOCATABLE AREA.

Figure 2-2. Allocatable Core Layout

Request priority levels 1 through 3 are restricted to the job processor after initialization. Level 0 is used for core swapping. During initialization of the system, the system directory request priorities are also all 0, making the entire allocatable area available. The user process programs must avoid use of request priorities 0 through 3.

NOTE

The request priority determines core allocation and queuing to receive space; the completion priority determines the execution (scheduling) priority of the program just loaded.

When there is no space available for the specified request priority, allocation is not attempted unless the specified priority is greater than the priority of any other space request in the stack. Instead, the request is threaded on a priority basis with others waiting for space. The request priority governs not only the area that is available to the program, but also the order on the space allocation thread. The thread is serviced on a first-in, first-out basis, within priority. If the only areas defined in a given system are 1 to 4, a priority 6 SPACE request is serviced prior to a priority 4 request, even though both requests use area 4.

When storage is released, the processor attempts to allocate the released area to the top request in the queue. The first address of an assigned block is stored in the fourth word of the SPACE request parameter list and is passed to the completion address in Q. The core allocator is used when making a SPACE request and is unavailable to the unprotected programs.

Caution should be exercised by programs that make additional SPACE requests for storage in allocatable core. If the requesting program occupies the space needed to satisfy the request, the request can never be completed.

PARTITIONED CORE

The area specified as partitioned core is defined in the SYSDAT program in a table of starting locations for the partitions (PARTBL). There may be from one to 16 partitions defined, with partition 0 as the first partition. There are always 16 entries in PARTBL, with negative zero indicating unused partitions. The number of the last partition available in a system is equated to LSTPRT. The last location of partitioned core plus one is contained in LSTLOC. There may also be a seventeenth partition: this is unprotected core when it is located at the top of core. Partitioned core must start at the beginning of part 1, at a location of 8000_{16} or less in core, but may end anyplace above the starting address. Figure 2-3 shows two possible partitioned core allocations.

Programs that execute in partitioned core reside on mass memory and are initiated by scheduling their ordinal numbers. Such a program, which may require one or more partitions, always executes in the same partitions. Since partitions may be in the upper bank, all monitor requests must have the D-bit set and all address parameters specified as 16-bit absolute.

Besides the partition table in SYSDAT, there is a table to indicate how many partitions are assigned to a single block (USE) and a table of tops of threads (THDS) for the partitions. Each partition has a thread associated with it.

There are two priority systems at work in the allocation of a partition. The first uses the request priority to place the request on a partition's thread. The second priority results from the direction of the scan of the thread tops. Since the scan for requests begins on the thread of partition 0, a low priority request for partitions 0 and 1 is processed before a higher priority request for partition 1.

Requests for core are threaded and processed unless the amount of core requested can never be available (for example, the core requested extends beyond the end of partitioned core). For this error, Q is set to 0 upon entry to the user's completion address.

Partitions 0 through 15 are always contiguous. When unprotected core is in part 1, is it always partition 16. It may not necessarily be contiguous to partition 15, as shown in figure 2-3.

Protected programs absolutized to partition 16 must run at a completion priority higher than 2 and, when scheduled, cause a memory swap if job processing is active (section 3).

CYBER 18 EXTENDED MEMORY

CYBER 18 systems may have a main memory larger than 65,535 words. MSOS supports this extended memory, which can only be accessed via a hardware page file, as a mass storage compatible peripheral device. MSOS READ, WRITE, FREAD, FWRITE, and MOTION requests to this device are handled the same as on mass storage equipment.

UNPROTECTED/PROTECTED COMMUNICATION

Unprotected programs may require entry to programs in protected core. For example, to save space, floating-point routines in protected core may be made available to programs in unprotected core. However, the protected core programs must be re-entrant and adhere to the following sequential rules:

1. The entry point of each program to be entered from unprotected core must be unprotected.
2. The entry point must be in the table of presets.
3. The program must be entered by an RTJ instruction.
4. An IIN instruction must immediately follow the entry point.
5. The called program must use volatile storage for intermediate results; interrupts may then be enabled.
6. The program must check the parameters passed from unprotected programs to prevent fatal errors (such as storing in protected core).
7. When the program exits it must release all requested volatile storage.

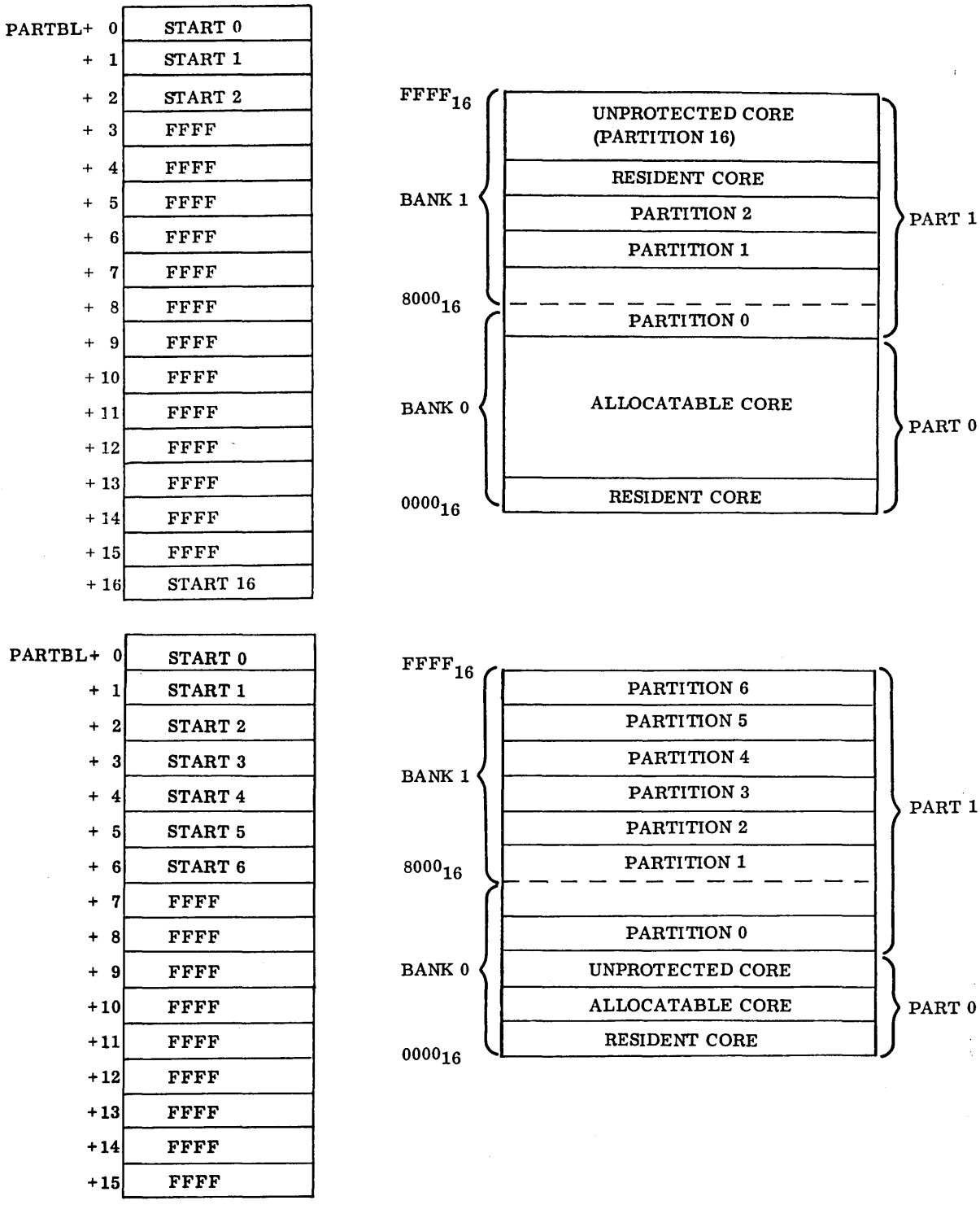


Figure 2-3. Partitioned Core Layout

UNPROTECTED ENTRY POINTS

Programs in protected core that are entered from unprotected core must have an available return location. An unprotected entry point entered by an RTJ provides this facility. Unprotected entry points are set up during system initialization through a table of entry point names (table of presets) in ASCII code which is assembled in the SYSDAT program. The RESTRT module in SPACE uses this table to find locations and to clear their protect bits. The table remains in core and is available to the relocating loader as preset entry points, with locations available to unprotected programs that declare the entry points as externals. The locations and length of the table are contained in the communications region (appendix B) for availability to the relocating loader.

CAUTION

Maximum system protection is provided when protected programs are entered from and returned to unprotected core. Programs entered from unprotected core should save their entry point addresses in volatile storage before enabling interrupts to prevent erroneous entries. Programs entered from unprotected core should not return to the job without checking to ensure that the job has not been aborted. If the job has been aborted, exit should be made to the dispatcher. Entry point JBCNFG in TRVEC is nonzero if the job has been cancelled.

Entries from unprotected core result in protect violations, with NIPROC transferring control to the protect processor which recognizes the violation as follows:

- Violation occurs at a location preceded by IIN
- The location preceding IIN is unprotected
- The location preceding IIN is filled with an address that follows an unprotected return jump to the unprotected entry point

When a protect violation is recognized, execution is resumed at IIN.

PROTECTED CORE-RESIDENT ENTRY POINT LINKAGE

At intialization, entry points for all protected core-resident programs are stored in either the CREP or CREP1 tables. These entry points are used by LIBEDT or the job processor to link a program to protected core-resident programs. The CREP table contains the entry point for all protected programs that execute in part 0 and the CREP1 table provides the linkage for all part 1 protected programs.

When LIBEDT encounters undefined external symbols while processing an *M statement for a part 0 program, it instructs the loader to attempt to satisfy them by linking to CREP and CREP1 tables.

In an *A LIBEDT statement that replaces a partition core program, linkage is attempted first to the CREP1 table and then to the CREP table (if required). The *P LIBEDT statement allows linkage in the order the user desires.

When the job processor encounters undefined external symbols after an *X or an *LGO statement has been processed, the operator types an *E statement to instruct the loader to attempt to satisfy the undefined external symbols.

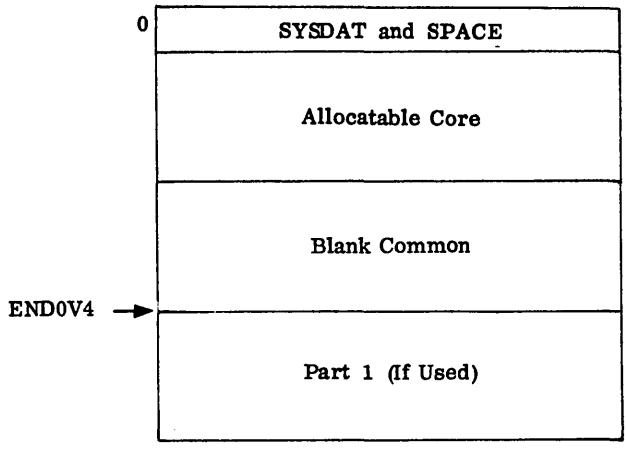
SYSTEM COMMON ORGANIZATION

MSOS provides for use of both blank and labeled common areas. Blank common cannot be preset with data. Labeled common can be preset with a DAT/ORG sequence in assembly language or a block data subprogram in FORTRAN. The system handling of common differs for protected and unprotected programs.

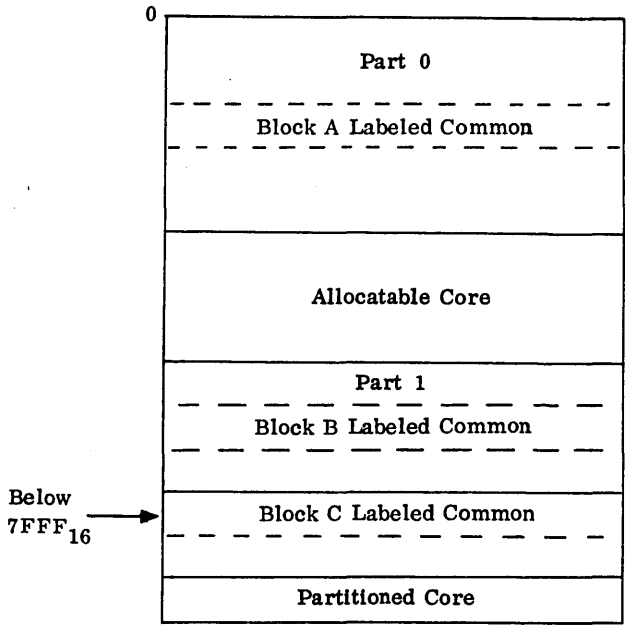
PROTECTED COMMON

For protected programs, one blank common area can be reserved in the system. This area must be assigned during system initialization and is restricted in size to the common block declaration of the first program declaring common. All programs subsequently loaded that declare common refer to this blank common block.

The common block is located in memory following allocatable core, with the highest location equal to the value of END0V4.



Labeled common can be located in one or more memory blocks, but must be core-resident (part 0 or 1) and be below $7FFF_{16}$. Labeled common blocks may not be part of mass memory system directory program groups, but must refer to a core-resident labeled common area. Labeled common blocks are created during system initialization with the *D control statement (section 6).

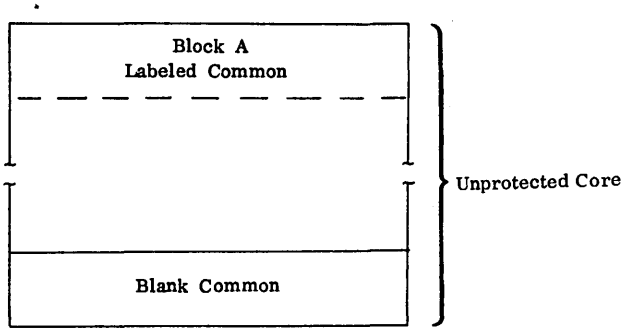


NOTE

All programs loaded by the initializer declaring blank common reference system blank common. All programs loaded by the initializer declaring labeled common (data) reference the last labeled common block loaded. Programs that run in partitioned core loaded under an *MP initializer command refer to system blank common. Through the use of LIBEDT (section 13), programs may be loaded via an *A command that has labeled common declared with the partition.

UNPROTECTED COMMON

For unprotected programs, one blank common area and one labeled common area may be specified. These areas are different than those allocated for protected programs. The common areas specified are assigned in unprotected core with the program(s) loaded. Blank common is defined at the top of unprotected core.



ENTRY FOR REQUEST

Requests within a program cause the monitor to perform operations such as reading, writing, loading, and program scheduling. A request may be written in assembly language as a macro instruction; for example:

```
FREAD     lu, c, s, n, m, rp, cp, a, x, d
```

Where: FREAD is the type of request

lu, c, s, n, m, rp, cp, a, x, d is the parameter list of request

The macro assembler converts the macro request into a calling sequence; for example:

```
Program
   .
   .
   .
RTJ- (F4₁₆)          Entry point for requests (NMONI)
parameters†          Parameters including constant
   .                 values not in the request param-
   .                 eter list (these may be supplied
   .                 by the macro assembler for this
                     macro).
parameters†
Program              Usual re-entry point for
   .                 continuation after threading the
   .                 request.
   .
```

At times the monitor has one or more other requests for the same type of operation (several requests to write or read data using the same disk). In this case, the new request cannot be executed immediately. Instead, the new request is queued for later execution. Three types of queuing are used by MSOS:

- I/O requests. These include READ and WRITE (formatted or unformatted), MOTION, and GETFILE requests. The logical unit that must perform the I/O request keeps a thread of queued requests. Entries in the thread are either the actual calling sequence embedded in the requesting program or, in the case of unprotected programs, a copy of the calling sequence kept in protected core to prevent request modification after the request has been error checked and threaded. Each item in the thread contains a pointer to the next calling sequence in the thread. When the I/O driver for this logical unit completes a request, it checks the thread. If another request is queued, processing for that request starts before the driver releases control. By this means, a driver has the ability to process every request in queue on a priority basis. During this period new requests can be inserted in the queue.

- Scheduling requests. These requests include both immediate scheduling and delayed scheduling (timer) requests. Each of these requests is moved to a scheduler stack (part of SYSDAT) and threaded there. Six separate threads coexist in the scheduler stack: one for scheduler requests, four for timer requests, and one for empty slots in the stack. The empty stack is checked by the monitor to find the next entry that can be used for a request. The dispatcher works requests off the scheduler stack, ensuring that every request in the stack is processed (except timer requests that have not yet become immediate requests) on a priority basis.

- Memory allocation requests. These requests include space assignment and release and partitioned core assignment. The calling sequence for the request is threaded in place. If no space is currently available to satisfy the request, it is threaded. Within request priority, space is allocated on a FIFO basis. Whenever a release request is completed, the space request thread is checked. If sufficient space exists for the first threaded request, the space is assigned.

Monitor requests must conform to the type of addressing required for the particular software configuration. In all cases the basic rule to observe is to set the d-bit for requests that reference or are made from part 1. Setting the d-bit forces all such request parameters to be absolute. This bit may also be used for part 0 requests if the request parameters are absolute.

Parameter lists, which execute and reference only in the lower bank, do not require the d-bit to be set and can therefore have indirect parameters in their parameter lists. Run-anywhere programs, if in part 0, may also use part 1 requests if they absolutize their request parameters. This is practical in any system program that runs in either allocatable or partitioned core, or unprotected or core-resident programs that may be required to run in the upper bank on another system.

The CYBER 18/1700 MSOS provides 19 monitor requests; each has an entry point Txx, where xx is the request code (1 to 19). Refer to table 3-1.

The numerical part of each name corresponds to the value of an index to a table of requests. All request processors are entered with the location of the request parameter list in the A register and exited by a jump to the request exit entry point. Request codes 20 to 25 are reserved for future system use.

†Parameter list of calling sequence

TABLE 3-1. MONITOR REQUEST TABLE

Request Code†	Request Mnemonic	Request Type	Program Type Usage	Program Handling Request	Can Be Queued?
0	–	System directory read	Monitor only	RW	Yes
1	READ	Normal read	All	RW	Yes
2	WRITE	Normal write	All	RW	Yes
3	STATUS	I/O request status	Unprotected	T3	No
4	FREAD	Formatted read	All	RW	Yes
5	EXIT	Unprotected exit	Unprotected	T5	No
6	FWRITE	Formatted write	All	RW	Yes
7	LOADER	Relocatable binary loader	Unprotected	T7	No
8	TIMER	Schedule program with delay	All	TMINT	Yes
9	SCHDLE	Schedule program	All	NDISP/ RDISP	Yes
10	SPACE	Allocate core	Protected	SPACE	Yes
11	CORE	Unprotected core bounds	Unprotected	T11	No
12	RELEAS	Release core	Protected	DCORE	No
13	GTFILE	Access permanent files in the program library	Unprotected	T13	Yes
14	MOTION	Tape motion	All	T14	Yes
15	TIMPT1	Schedule directory program with delay	Protected	TMINT	Yes
16	INDIR	Indirect (use another parameter list)	All	T16	Yes
17	PTNCOR	Allocate partitioned core	Protected	NDISP/ RDISP	Yes
18	SYSCHD	Schedule directory program	Protected	NDISP/ RDISP	Yes
19	DISCHD/ ENSCHD	Disable/enable system directory scheduling	Protected	NDISP/ RDISP	No
20 21 22 23 24 25		Reserved for future use			

THREADING

THREADING IN PLACE

Certain monitor requests provide a thread word within the parameter list to allow requests to be threaded by priority while waiting to be processed. Requests from protected programs are threaded in place using the thread word of the calling sequence. Unprotected programs cannot depend on this type of in-place threading since an unprotected program could change the request parameters after threading. Therefore, up to five unprotected requests can be moved into protected core and threaded there. The number of requests is arbitrarily set at five to limit the amount of protected space needed to store these requests.

Requests are threaded on a first-in-first-out (FIFO) basis within each priority level. The logical unit table (LOG2) holds a pointer to the first unprocessed parameter list calling sequence. As soon as this request is started, the LOG2 pointer is changed to the address given by the thread word. Then the thread word is set to $FFFF_{16}$, indicating that the request is being processed. The last request in the thread always has its thread pointer set to $FFFF_{16}$. When a request is completed, the thread word is set to $+0_{16}$.

In the example of figure 3-1, five requests are shown. These could be in the same or different programs but all are for the same logical unit (lu). Also, the requests could be for different types of operations; e.g., READ, WRITE, FREAD, and FWRITE, all for the same logical unit.

P,Q,R,S,T	The addresses of the five parameter lists
N	Logical unit. Each request specifies lu N.
E	The physical device table address for the device corresponding to logical unit N (appendix C)
LOG2	The logical unit table (appendix C)

Parameter lists	P	Q	R	S	T
Completion priority	8	1	8	4	4
Request priority	3	9	0	4	1

Threaded by this priority

In this example, request T has a lower priority than P, Q, and S. However, if request T is made before P, Q, and S, request T will be started and completed before the others are processed. Assume that request S is not active at this time.

When a request has been threaded, control returns to the address following the last word of the calling sequence, if no request with a higher priority is waiting to run. The user can, therefore, continue processing while the input/output requested is in progress. If the program cannot continue until completion of the input/output request, it should exit to the dispatcher and allow other programs to run until the completion address is scheduled. Unless the program is unprotected, the user program should not loop while waiting for completion by testing the request thread for zero.

CAUTION

Requesters that loop on the request thread at priorities 2 and above can cause serious interference with monitor functions and inhibit system real-time performance.

Completion priority is meaningless if no completion address is specified.

NOTE

Users should not assume that the contents of the A, Q, or I registers are returned to the user in their original state following MSOS requests. The contents of these registers may vary from request to request; within requests, the contents may vary from foreground to background.

THREADING IN STACKS

The parameter lists of scheduler and timer requests are copied into the scheduler stack as segments of one of the five active threads: requests for immediate execution (SCHDLE), or requests delayed for n counts, tenths of seconds, seconds, or minutes (four separate timer threads). SCHDLE requests are threaded by priority on an FIFO basis; TIMER requests are threaded by time until activation in the appropriate timer thread. When a TIMER request matures, a SCHDLE request rethreads it to the SCHDLE thread. When the dispatcher gains control, it checks both the scheduler and interrupt stacks. If the scheduler stack has the highest priority program awaiting execution, the dispatcher executes the first entry in the SCHDLE thread. The space occupied by that request's parameter list is returned to the thread of unused spaces. As described in section 2, only SCHDLE requests that are at the same or lower priority than that of the calling program are stacked; higher priority programs are executed immediately.

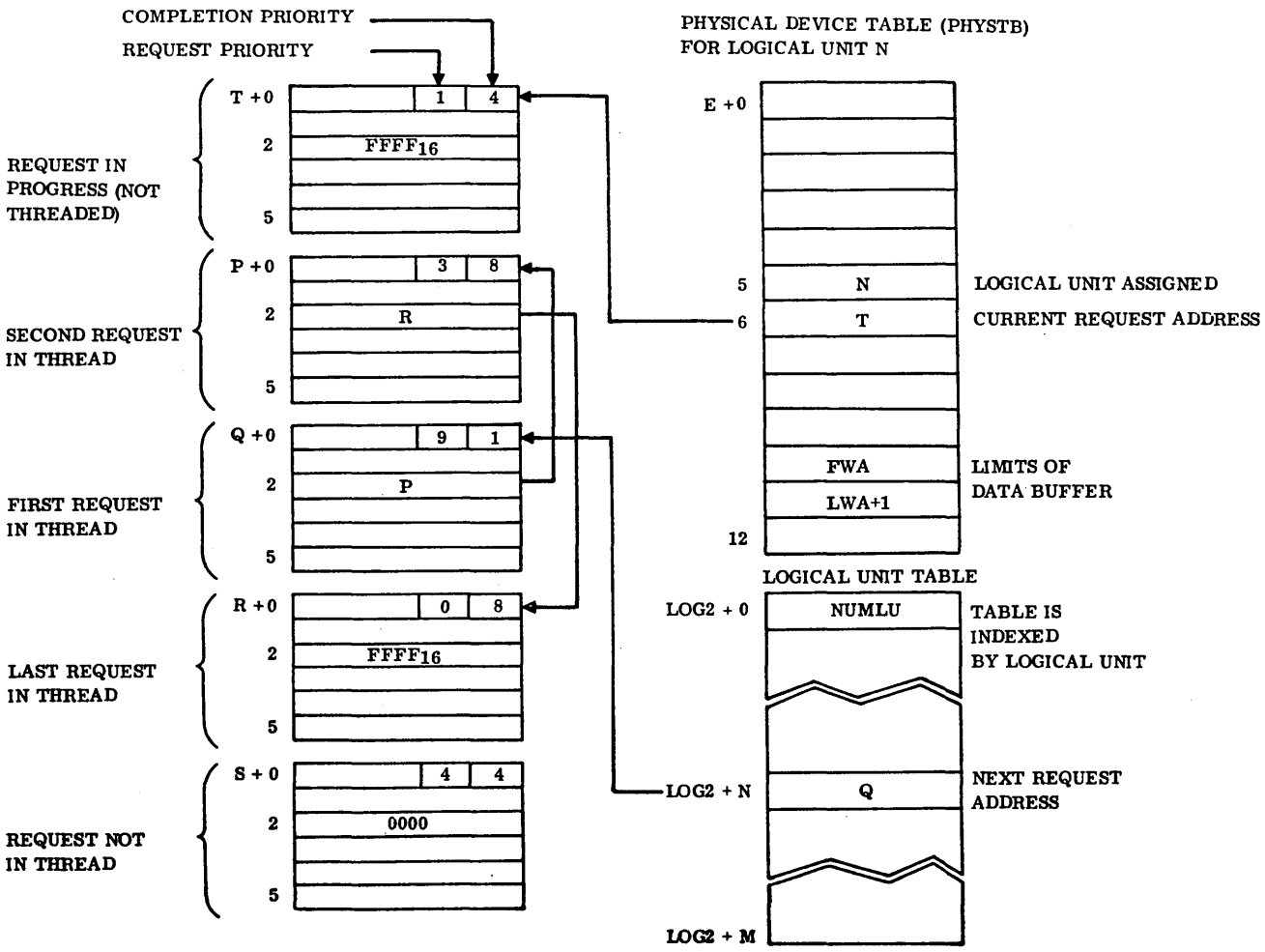
When the scheduler stack is full, new requests are rejected until space is available.

To notify the requesting program that the request has been threaded in the stack, the sign bit of Q is set:

Bit 15	0	Request is accepted
	1	Request is rejected
Bits 14 through 0		Unchanged

When scheduler requests are made for system directory programs, only one request may be processed at a time (allocate core, read program from mass memory, schedule program). Subsequent program requests are rejected (bit 15 of the Q register is set to 1) until the initial request is completed. Timer requests for system directory programs that expire when a previous scheduler request is in process are automatically retried on the next count interrupt until successfully scheduled.

Scheduler requests made for system directory programs that have been disabled (i.e., those operated on by DISCHD request) are rejected with bit 15 of the Q register set to 1.



NOTES:
1. LOCATIONS P, Q, R, S, AND T ARE IN THE USER PROGRAM, AND ARE STARTING POINTS OF THE PARAMETER LISTS IN THE CALLING SEQUENCES.
2. REQUEST T WAS STARTED BEFORE P, Q, AND R WERE THREADED.
3. REQUEST S IS EITHER COMPLETED OR HAS NOT YET BEEN MADE.

Figure 3-1. Threading Example

REQUEST DESCRIPTIONS

Some of the 19 requests supplied with the CYBER 18/1700 MSOS are applicable to both protected and unprotected programs, and others are unique to either protected or unprotected programs.

PROTECTED AND UNPROTECTED PROGRAM REQUESTS

The requests allowed for both protected and unprotected programs are:

Request	Request Code	Description
READ	1	Read record
WRITE	2	Write record
FREAD	4	Read format record
FWRITE	6	Write format record
INDIR	16	Indirect execute request Indirect request to part 1
TIMER	8	Schedule program on priority basis after specified time delay elapses
SCHDLE	9	Schedule program on priority basis
MOTION	14	Control peripheral device motion

READ/FREAD/WRITE/FWRITE

READ/WRITE instructions transfer data between the specified input/output device and core. The word count specified in the request determines the end of the transfer.

FREAD/FWRITE requests read/write records in a specific format for each device.

The macro format for READ/WRITE/FREAD/FWRITE requests (1,2,4,6) is shown below:

READ
FREAD
WRITE
FWRITE } lu, c, s, n, m, rp, cp, a, x, d

Where: lu is the logical unit.

c is the completion address.

s is the starting address.

n is the number of words to transfer.

m is the mode.

rp is the request priority.

cp is the completion priority.

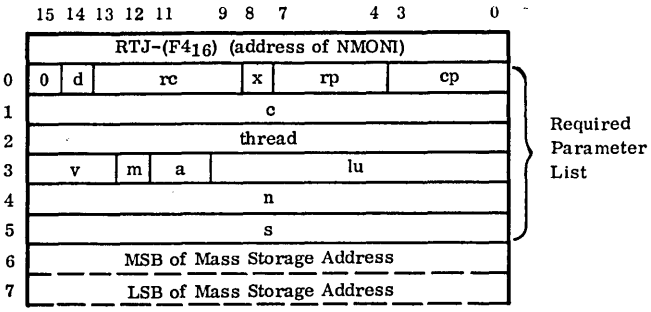
a is the absolute/indirect indicator for the logical unit.

x is the relative/indirect indicator (affects parameters c, s, and n).

d is the part 1 request indicator (absolute parameter addresses).

A detailed description of each of the above parameters follows after the calling sequence generated by the macro.

The request codes are 1 (READ), 2 (WRITE), 4 (FREAD), and 6 (FWRITE). The calling sequence generated by the macro is as follows:



The field descriptions for the calling sequence generated by READ/WRITE macros are:

rc — The request code

thread — The thread location used to point to the next entry or the threaded list. Each logical unit has its own thread; completion of one request causes the logical unit driver to inspect the thread, if any, and to start the transfer specified by the first entry in the thread.

v — Error code. This code is passed to the completion address in bits 15 through 13 of Q. This code is also set into the calling sequence at request completion.

Detailed parameter descriptions for the requests are:

lu is the logical unit; an index to the LOG1A table of physical device table addresses (appendix C) may be modified by parameter a.

c is the completion address of the main memory location to which control is transferred when an I/O operation is completed. If omitted, no

completion routine is scheduled and control is returned to the interrupted program. If given in (c) format, it is an index to the system library directory; (c) indicates the program to be executed upon completion of the requested I/O operation. Use of the (c) option by unprotected programs results in job termination.

Completion routines are operated by threading the I/O requests on the scheduler thread. A three-bit code in the v field of the fourth word of the request indicates completion status:

15	14	13	Description
0	0	0	No error condition detected by driver; number of words requested was read or written; device not ready
0	0	1	No error; requested number of words read or written; device ready
0	1	0	No error condition detected by driver; fewer words read than requested; device not ready
0	1	1	No error; fewer words read than requested; device ready
1	0	0	Error condition; requested words read; device not ready
1	0	1	Error condition detected by driver; number of words requested are read or written; device ready
1	1	0	Error condition and/or end-of-file detected; fewer words read than requested; device not ready
1	1	1	Error condition and/or end-of-file detected; fewer words read than requested; device ready

When control is returned to the completion address, these bits are set in similar positions in Q. if less than n words were transferred on a read operation, the end of buffered data (location following the last word transferred) is placed in the last word of the user's buffer.

For all devices having an end-of-file (EOF) indication, EOF can be verified by checking bit 11 of word 12 in the physical device table.

s is the starting address; the address of the data buffer to be transferred (see parameter x).

n is the number of words to be transferred.

(n) Number of words to be transferred is determined by parameter x.

0 The minimum information is transferred (one word or one character), depending on the device.

NOTE

For FREAD and FWRITE, n cannot be zero. Some devices signal zero words as an illegal request.

m is the mode; determines the operating condition (binary/ASCII) of a driver.

Macro

A Data is converted from ASCII to external form for output; from external form to ASCII for input.

B Data is transferred as it appears in core or on an I/O device.

Coding

0 Binary

1 ASCII

rp is the request priority (15 through 0, with 0 as the lowest) with respect to other requests for this device. This request establishes the order in the I/O device queue. It is automatically zero for unprotected requests.

cp is the completion priority (15 through 0); the level at which the sequence of the code specified by parameter c is executed. It is automatically one for unprotected requests. (see Scheduling Tasks by Priorities and Interrupts, section 2, for an explanation of priority levels.)

a is the absolute/indicator for the logical unit.

Macro

blank lu specifies the logical unit number.

R lu is a signed increment $-1FF_{16} = lu \leq 1FF_{16}$, which is added to the address of the first word of the parameter list to obtain the core location containing the logical unit number.

I lu is the address of the core location containing the logical unit number ($lu \leq 3FF_{16}$).

Coding

0	lu is a logical unit number.
1	lu is a signed increment ($\pm 1FF_{16}$); not allowed if d = 1.
2	lu is a core address containing the logical unit number.

x is the relative/indirect indicator; this parameter affects parameters c, s, and n as shown here. Becuase of the wrap-around feature, computed addresses may be before or after the parameter list.

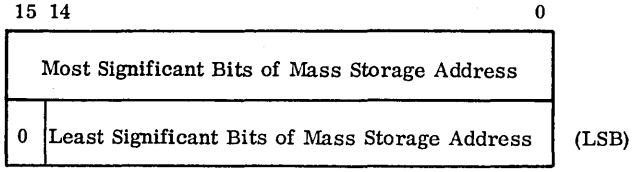
(c) is indirect	x is meaningless and c represents an index to the system directory.
0 or blank and c is direct	c is the completion address.
0 or blank and s is direct	s is the starting address. If the request is on mass memory, the mass memory address will be words 6 and 7 of the calling sequence.
0 or blank and (s) is indirect	s is a core location that contains the starting address. If the request is on mass memory, the mass memory address follows the core location that contains the starting address.†
≠ 0 or not blank and c is direct	c is a positive increment that is added to the address of the first word of the parameter list to form the completion address.
≠ 0 or not blank and s is direct	s is a positive increment added to the address of the first word of the parameter list to form the starting address. If the request is on mass memory, the mass memory address will be words 6 and 7 of the calling sequence.
≠ 0 or not blank and (s) is indirect	s is a positive increment added to the address of the parameter list to form the address of a location containing another positive increment. If the request is on mass memory, the location containing the second increment is immediately followed by two words which contain the mass memory address.
n is direct	x is meaningless and n is the length of the block to be transferred.
x is 0 or blank and (n) is indirect	n is the core location containing the block size.
x is ≠ 0 or ≠ blank and (n) is indirect	n is a positive increment added to the address of the first word of the parameter list to obtain the location containing the block size.

d is the part 1 request indicator; this parameter indicates that the request requires the use of 16-bit address arithmetic.

0 or blank	Preceding description of parameter applies	
1	x	is ignored.
	n	is the number of words
	c and s	are 16-bit absolute addresses.
	lu	is processed the same as d = 0.
	a	cannot be set to R.

MSA is the mass storage address. It is not supplied by the macro. The two words must be supplied by the program.

The memory address format is:



The mass storage address specifies a mass memory word address (READ/WRITE) or a mass memory sector (96-word size) address (FREAD/FWRITE).

NOTE

Following threading of the request, control is returned to the program at the continuation address. This address is the location following the parameter list. (Note that the mass storage address may be the last two words of the parameter list.)

INDIR

The INDIR request (16) allows indirect execution of any other request. Use of INDIR allows one call (e.g., FREAD) to be used several places in a program without needing to recompute certain parameters. The request to be used is specified by the address of the first word of the other request calling sequence:

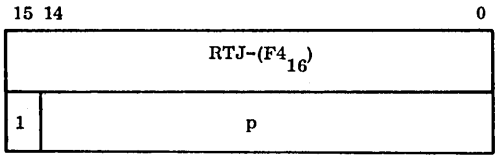
INDIR p,i

†If bit 15 is set for (n) or (s), incrementing continues indirect until bit 15 is not set.

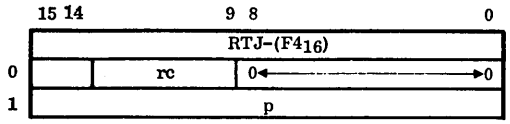
Where: p is the address of the first word of the parameter list in the calling sequence of any other request; p must not be enclosed in parentheses.

i is the indicator for 15- or 16-bit core addressing:

0 or blank	Part 0 (15-bit addressing) contains the calling sequence and INDIR has no request code. The calling sequence generated by the macro is:



	This form is only useful when the address of the request to be executed is at an address below 8000_{16}.
1	INDIR requires 16-bit core addressing (part 1 of the CPU). The calling sequence generated by the macro is:



TIMER

A TIMER request (8) is a delayed SCHDLE request. Through the user of TIMER, a SCHDLE request is made after the specified time elapses. The macro format is:

TIMER c,p,x,t,u,d

Where: c is the completion address to be executed.

p is the priority level of the program. p should be less than or equal to the priority level of TIMINT. The use of priority levels above this level allows the possiblity of losing an entry for the same timer thread as a result of a timer interrupt.

x is the relative/indirect indicator.

(c) is indirect	x is meaningless and c represents an index to the system directory. The entry referred to by the index specifies the program. This feature is not available to unprotected programs and programs running in part 1.
0 or blank and c is direct	c is the location to be executed.
≠ 0 or not blank and c is direct	c is a positive increment added to the address of the first word of the parameter list to obtain the execution location. Because of memory wrap-around, the execution location may be before or after the TIMER request.

t is the time delay.

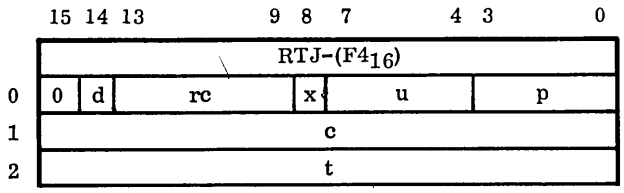
u is the units of delay; this parameter determines the units in which the time delay t is measured.

0 or blank	t is the basic unit of the timing device (counts).
1	t is measured in tenths of a second.
2	t is measured in seconds.
3	t is measured in minutes.

d is the part 1 request indicator (absolute request parameters).

0 or blank	All parameters operate as described above.
1	x is ignored and c is a 16-bit absolute address; all other parameters operate as described above.

The request code is 8 and the calling sequence generated by the macro is as follows:



TIMER requests are stacked in the scheduler list, but are not threaded to the SCHDLE requests. Instead, they are threaded together on the basis of time until activation. These threads are checked by the timer routine periodically. When the delay for a TIMER request has elapsed, a SCHDLE request is made with Q equal to the contents of location $E8_{16}$ (the core clock counter) at the time the SCHDLE request was made. This former timer request, if it is threaded at all (see SCHDLE, below), is then threaded to the SCHDLE requests on a FIFO-within-request-priority basis. An external parameter in SYSDAT specifies the number of simultaneous TIMER expirations permitted to prevent loss of interrupts if time is insufficient to process the number of TIMER requests expiring at one time.

SCHDLE

Programs are queued on a priority basis through the use of the SCHDLE request (9). A program requested by SCHDLE is executed only when it is the oldest waiting task with the highest priority. All programs specified by SCHDLE requests are entered by a simple jump and exited by a jump to entry point DISP (protected) or by an EXIT request (unprotected). The value in the Q register is passed to the requested program on entry. The macro format is:

```
SCHDLE c,p,x,d
```

Where: c is the address to be executed as described under parameter x.

p is the priority level of the program; for unprotected programs, p is 1. If two programs are of equal priority, the one in progress is continued.

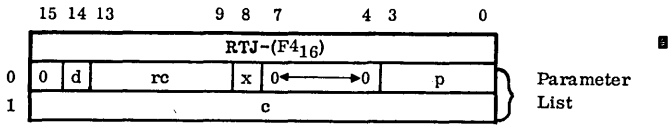
x is the relative/indirect indicator.

(c) is indirect	x is meaningless and c represents an index to the system directory. The entry referred to by the index specifies the program. This feature is not available to unprotected programs and cannot be used if running in part 1 by protected programs (refer to the SYSCHD request in this section).
0 or blank and c is direct	c is the location to be executed.
≠ 0 or not blank and c is direct	c is a positive increment added to the address of the first word of the parameter list to obtain the execution location. Because 15-bit arithmetic is used, the execution location may be before or after the SCHDLE request.
≠ 0 or not blank	(c) indirect is illegal.

d is the part 1 request indicator (absolute request parameters).

0 or blank	Parameters are processed as described above.
1	The x parameter is ignored and c is processed as a 16-bit absolute address.

The SCHDLE request code is 9 and the calling sequence generated by the macro is:



Example:

```
ENQ       1

SCHDLE    (COMP), 6
```

At system initialization, entry point COMP is associated with an index to the system directory. The program associated with the system directory index (referred to by COMP) is operated at priority level 6. On entry to that program, Q contains 0001_{16}. The program may be core-resident or mass-storage-resident, according to the system directory entry. If the request is written in the following manner, the core location associated with label COMP is executed at priority level 6 and is entered with 0001 in Q.

```
ENQ       1

SCHDLE    COMP, 6
```

If a new program is at a higher priority level than the current level, the request is not queued, but is immediately executed. If the requested program is mass-storage-resident, the scheduler request processor causes the allocation of core for this program and requests the transfer of the program from mass storage to core.

If the program priority level is less than or equal to the current level, the parameter list of the request is moved to the scheduler stack (SCHSTK) in SYSDAT, and is threaded by priority on a FIFO basis within the priority.

The queuing routines move the entries to SCHSTK, and the dispatcher removes entries and threads the returned space to the SCHSTK empty thread. When an input/output request is completed, the driver causes the completion routine to be executed by threading the input/output request to the scheduler list. This process avoids filling the list with input/output completion addresses.

MOTION

This request (14) is used to control motion and end-of-file processing. The macro format is:

MOTION lu,c,p_1,p_2,p_3,dy,rp,cp,a,x,d,m

Where: lu is the logical unit.

c is the completion address.

p_1,p_2,p_3 are the motion control parameters. Each of these results in a specific action, which is defined in table 3-2. Up to three motion commands may be defined in a MOTION request; they are executed in the sequence p_1 ,p_2 ,p_3 . The first command with a value of zero terminates the request.

dy is the density parameter.

0	No change	External rejects will result when an illegal density selection is attempted.†
1	800 frames per inch	
2	556 frames per inch	
3	200 frames per inch	
4	1600 frames per inch	

rp is the request priority.

cp is the completion priority.

a is the absolute/indirect indicator for the logical unit.

x is related only to the completion address.

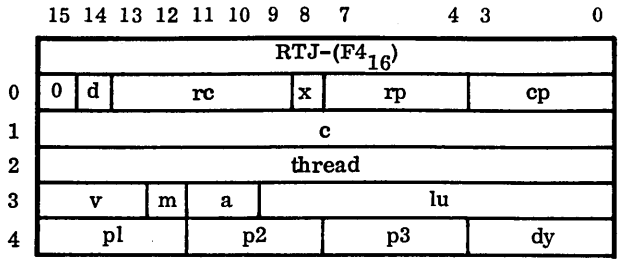
d is set to 0 All parameters are processed as described.

1 A part 1 request is indicated (c is a 16-bit absolute address and must not equal R for the a parameter).

m is the mode.

Mode	Code	
A	1	ASCII
B	0	Binary

The MOTION control request code is 14 and the calling sequence generated by the macro is as follows.



Where: rc is the request code.

thread is the thread location used to point to the next entry or the threaded list.

v is the error code setting.

One MOTION control can be repeated for magnetic tape. In this case the macro request is as follows:

MOTION lu,c,r,p,n,0,rp,cp,a,x,d,m

Where: r is the repeat function indicator, which must equal R in the a parameter.

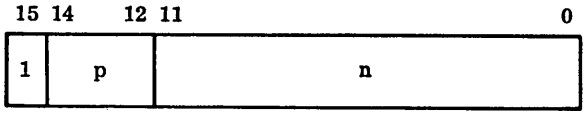
p is the motion code.

n is the number of times to be executed, not to exceed 4095.

0 is a null parameter.

All of the parameters are the same as in the preceding MOTION request except for r, p, n, and 0, which replace p_1, p_2, p_3, and dy.

The coding sequence generated is the same as above except for the last word, which is generated as follows:



Where: 1 indicates that the request can be repeated.

The following macros can also be used for MOTION requests; each macro can perform only one MOTION request. (Refer to table 3-2 to determine the action taken by the driver.)

BSR*	lu,a,n,c,p
EOF*	lu,a,n,c,p
REW*	lu,a,n,c,p
UNL*	lu,a,n,c,p

†In addition to the attempt to set a density which is not legal for a unit (e.g., 200/556 frames per inch on a 609), the drivers do not allow a density change if the unit is not at load point.

TABLE 3-2. MSOS DRIVER ACTION FOR MOTION REQUEST PARAMETERS P_1, P_2, P_3

Code	Description	MT	CR	CP	LP	TTY	PTR	PTP	MSD	PTD	CD†
0	First zero terminates processing the request	X	X	X	X	X	X	X	X	X	X
1	Backspace one record	X								X	X
	Do nothing		X	X	X	X	X	X	X		
2	Write one end-of-file mark	X								X	X
	Punch one end-of-file mark			X				X			
	Page eject; reset line count				X	X					
	Punch leader							X			
	Do nothing		X				X		X		
3	Rewind to loadpoint	X									X
	Set pointer to start of tape									X	
	Do nothing		X	X	X	X	X	X	X		
4	Rewind and unload; terminates request	X									X
	Terminates processing the request					X	X	X			
	Sequence count goes to zero; terminates request		X	X							
	Reset line count; terminates request				X						
	Set pointer to start of tape; terminate this request									X	
	Do nothing								X		
5	Skip one file forward	X					X			X	X
	Slew cards to end-of-file		X								
	Do nothing			X	X	X		X	X		
6	Skip one file backward	X								X	X
	Do nothing		X	X	X	X	X	X	X		
7	Advance one record	X								X	X
	Do nothing		X	X	X	X	X	X	X		

KEY:

MT	Magnetic tape	TTY	Teletypewriter	PTD	Pseudo tape driver
CR	Card reader	PTR	Paper tape reader	CD	COSY driver
CP	Card punch	PTP	Paper tape punch		
LP	Line printer	MSD	Mass storage driver		

†Assumes use of the magnetic tape physical device; for detailed information refer to the Peripheral Drivers Reference Manual.

ADF* lu,a,n,c,p

BSF* lu,a,n,c,p

ADR* lu,a,n,c,p

Where: * specifies a relative completion address. If left blank, there is absolute completion. (The macro computes the relative address constant.)

lu is the logical unit number of the device.

a is the absolute/indirect/relative indicator for the logical unit.

blank lu is the actual logical unit number.

R lu is a signed increment ($-1FF_{16} \leq lu \leq 1FF_{16}$) added to the address of the first word address of the parameter list to obtain the address of a location containing the actual logical unit number.

I lu is a core address (0 to $3FF_{16}$) that contains the logical unit number.

n is the number of iterations. If blank, 1 is assumed (not to exceed 4095).

c is the completion address. If the macro call terminator is an *, completion is relative (only the label name is required). If the macro call terminator is a blank, the completion is absolute. If C is left blank, there is no completion.

p is the priority level; defines both the request and completion priority. If left blank, the priority is zero.

All parameters are optional and may be left blank, with the exception of lu.

Examples:

The following parameters are common to the four examples that follow.

NEXT is the completion address.

6 is the logical unit of the magnetic tape.

10 is the logical unit of the card punch.

MT is the program location containing a 6.

FA_{16} is the low core location containing the standard binary output device.

1. A backspace macro with the following: A relative location containing the logical unit number, backspace three records, a relative completion address, and a request and completion priority of 3.

   BSR* MT,R,3,NEXT,3

2. Same as the above, except that the completion address is absolute.

   BSR MT,R,3,NEXT,3

3. An end-of-file macro with the following: The actual logical unit number, write one end-of-file, zero completion, and a priority of 0.

   EOF 10

4. Same as above, except that the logical unit number is in a low core location.

   EOF FA_{16},I,,,0

PROTECTED PROGRAM REQUESTS

The following requests, which are available only to protected programs, are described in this section.

Request	Request Code	Description
SPACE	10	Allocate protected core for system library programs.
RELEAS	12	Release protected core used for system library programs.
DISCHD	19	Disable scheduled system directory programs.
ENSCHD	19	Enable disabled scheduled system directory programs.
SYSCHD	18	Part 1 system directory request
TIMPT1	15	Schedule system directory program from part 1 after time delay.
PTNCOR	17	Allocate one or more partitions for system library programs.

SPACE

Space in core must be allocated by the SPACE request processor to execute mass-storage-resident programs. If a request cannot be completed, it is threaded to the space allocator. Each RELEASE request causes the thread to be processed; space is assigned to the top entry of the thread, if possible, at that time. The macro format is:

SPACE n,c,rp,cp,x,d

Where: n is the number of words necessary (see x).

c is the completion address to which control is transferred when core space is allocated (see x).

rp is the request priority (with respect to other SPACE requests). If space is not available, requests are threaded together so that the oldest (highest priority) is filled first when space becomes available. This priority is also used as the index to the table LVLSTR to determine the starting address of allocatable core for the request priority. This has the effect of providing larger areas of core to SPACE requests with a higher priority level.

cp is the completion priority; the level at which the completion address is entered.

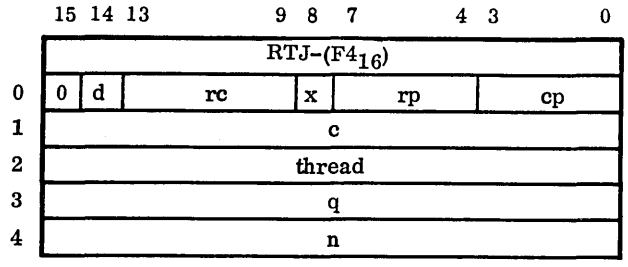
x is the relative/indirect indicator.

(c) is indirect	x has no meaning and c represents an index to the system directory. This parameter is not allowed if running in part 1.
n is direct	x has no meaning and n is the number of words.
0 or blank and c is direct	c is the completion address (jump to c is made).
0 or blank and (n) is indirect	n is the address of a location containing the number of words.
≠ 0 or not blank and c is direct	c is a positive increment added to the address of the first word of the parameter list to obtain the completion address. Because of memory wrap-around, this address can be before or after the parameter list even though c is positive.
≠ 0 or not blank and (n) is indirect	n is a positive increment added to the address of the first word of the parameter list to obtain the address of the location containing the number of words. Because of memory wrap-around, this location can be before or after the parameter list.

d is the part 1 request indicator.

0 or blank	All parameters function as described.
1	x is ignored; c and n are 16-bit absolute numbers.

The SPACE request code is 10 and the calling sequence generated by the macro is as follows:



Where: rc is the request code.

thread is the thread location used to point to the next entry or the threaded list. It must be set to zero initially and is reset to zero upon completion.

q is the address of the area allocated; it is in the Q register when control is given to the completion address, c. If allocation is impossible, Q is set negative.

RELEAS

The RELEAS request (12) is used to return to the system storage acquired by a SPACE request. After the space is released, the SPACE or PTNCOR request thread is checked to find if the first threaded entry can now be completed. The request macro format is:

RELEAS s,t,x,d

Where: s is the starting address of the block to be released (see parameter x). If this address is not the same as the address returned from a SPACE request, core space is not released; however, an error does not occur. If d = 1, then s is the 16-bit address of the beginning partition of partitioned core to be released.

t is the exit indicator.

0 or blank	RELEAS returns to requester at the location following the request parameter list.
≠ 0 or not blank	The RELEAS request processor jumps to the dispatcher.

x is the relative/indirect indicator.

s is direct	x is meaningless and s is the starting address of the block.
x is 0 or blank and (s) is indirect	s is an address of a core location containing the starting address of the block.
x is ≠ 0 or not blank and (s) is indirect	s is a positive increment added to the address of the first word of the parameter list to obtain the starting address of the block.

d is the part 1 request indicator.

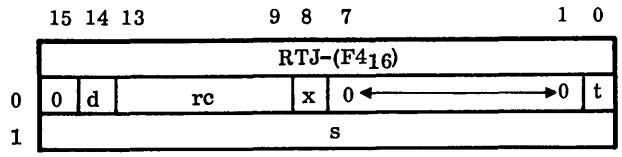
0	Allocatable core is released.
1	Partitions are to be released; s is the 16-bit address of the beginning partition of the area to be released. The same

number of partitions are released as were allocated (refer to PTNCOR), beginning at that partition. If several partitions are requested in a block, they must be released in a block. The contents of ($F7_{16}$) + 1 must be used to release the unprotected partition (refer to PTNCOR).

NOTE

Programs releasing the memory they occupy must use the exit indicator form: t is neither 0 nor blank.

The RELEAS request code is 12 and the calling sequence generated by the macro is:



When system programs are mass-storage-resident and scheduled via the system directory, core is allocated to them by a system SPACE or PTNCOR request.

Core must be released with a RELEAS request before exiting. To make core-resident and mass-storage-resident programs similar so that a program can be stored in either place (initialization with no change), all programs should attempt to release core upon exiting. Core occupied by core-resident programs is not released.

DISCHD

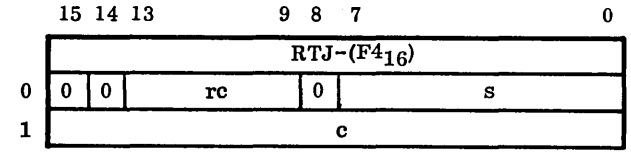
With the DISCHD request (19), the scheduling of specific system directory programs can be disabled for a period of time. The ability to schedule these programs can be restored by an enable request (ENSCHD).

The DISCHD request sets bit 15 to one of the first word of the system directory for the program to be disabled. This flag is checked whenever a system directory program is scheduled. The macro format is:

DISCHD c

Where: c is the index to the system directory.

The DISCHD request code is 19. The calling sequence generated is:



Where: rc is the request code.

s is set to FF_{16} for a disable request.

CAUTION

If a schedule request is rejected because the disable flag is set, a system directory program that may be scheduled using timer requests will have its request automatically repeated on each count interrupt after the delay has expired. The DISCHD request in this case creates increased system overhead in the timer request processor.

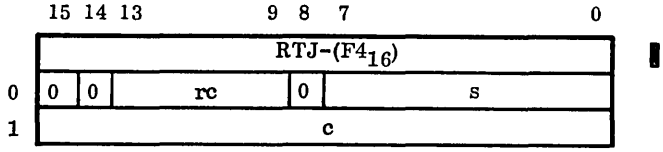
ENSCHD

The ENSCHD request (19) enables the scheduling of system directory programs after they have been disabled by a DISCHD request. The ENDSCHD request sets bit 15 of the first word of the system directory entry for the program to zero. The macro format is:

ENSCHD c

Where: c is the index to the system directory.

The ENSCHD request code is 19. The calling sequence generated is:



Where: rc is the request code.

s is set to 0 for an enable request.

SYSCHD

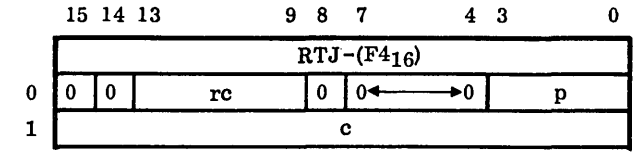
Protected programs which run in part 1 must use this request to schedule a system directory program. It may also be used for programs that run in part 0. The macro request format is:

SYSCHD c,p

Where: c is the index to the system directory. The entry referred to by the index specifies the program. Note that bit 15 of the c parameter is not set to one in this request form.

p is the priority level of the program.

The SYSCHD request code is 18 and the calling sequence generated by the macro is as follows.



TIMPT1

The part 1 TIMER (delayed scheduling) request must be used for scheduling system directory programs that are loaded in part 1. This request may also be used for part 0 programs. The calling sequence generated by the macro is as follows.

TIMER from Part 1

The macro format is:

TIMPT1 c,p,x,t,u

Where: c is the index to the system directory. Note that bit 15 of the c parameter is not set to one in this request form.

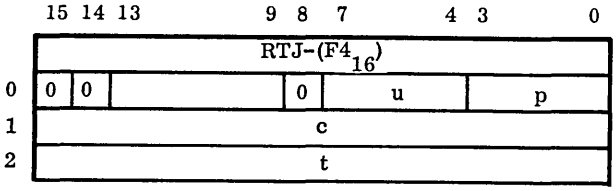
p is the priority level of the program (see p in TIMER request description for restrictions to the priority level).

x has no meaning.

t is the time delay.

u is the units of delay. This parameter determines the units in which the time delay is measured.

0 or blank	t is the basic unit of the timing device (counts).
1	t is measured in tenths of a second.
2	t is measured in seconds.
3	t is measured in minutes.



PTNCOR

The PTNCOR request (17) is used to allocate a block of partitioned core. The macro format is:

PTNCOR n,c,p,rp,cp,x,d

Where: n is the number of words in block to be allocated. This number determines how many partitions are allocated, but it does not need to be the exact length of the combined partitions (unaffected by x parameter).

c is the completion address, which is modified by x if d is not set.

p is the starting partition number; the number of the first partition in the block to be allocated (partitions are numbered zero through fifteen).

rp is the request priority. This priority governs where this request is threaded on the thread of partition p if more than one request is on the thread.

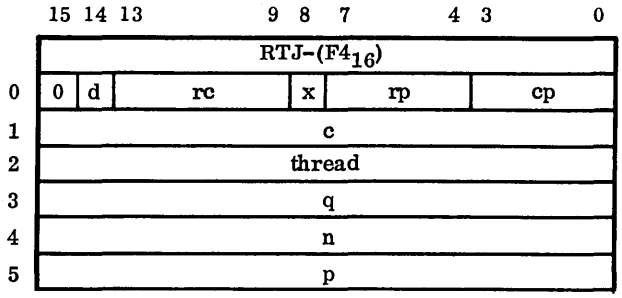
cp is the completion priority; the level at which the completion address is entered.

x is the relative/indirect indicator; x affects c as follows:

(c) is indirect	c represents an index to the system directory and x has no meaning.
x is 0 or blank and c is direct	c is the completion address and a jump to c is made.
x is ≠ 0 or not blank and c is direct	c is a positive increment added to the address of the first word of the parameter list to obtain the completion address. Because of 15-bit arithmetic, this address can be before or after the parameter list, even though c is positive.

d is the part 1 request indicator. x is ignored and c must be a 16-bit absolute address.

The request code for PTNCOR is 17 and the calling sequence generated by the macro is as follows:



Where: rc is the request code.

thread is the thread location used to point to the next entry on the threaded list. These calling sequences are threaded to the partitioned core allocator; the thread is checked whenever the RELEAS request releases partitioned core.

q is the address of the area allocated and is in the Q register when control is given to the completion address, c. If allocation is impossible, Q is set to zero.

UNPROTECTED PROGRAM REQUESTS

The following requests, which are only available to unprotected programs, are described in this section.

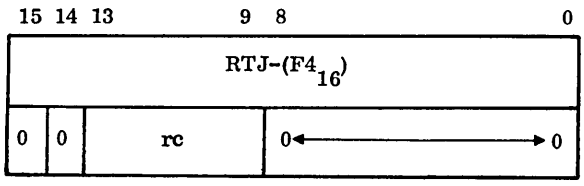
Request	Request Code	Description
CORE	11	Set or determine bounds of unprotected core.
LOADER	7	Operate relocatable binary loader.
GTFILE	13	Access permanent files in the program library.
STATUS	3	Determine the status of the input/output request.
EXIT	5	Exit from the unprotected program.

CORE

This request is used to set or determine the bounds of available unprotected core (that portion of unprotected core not occupied by a program or data for a job). If the A and Q registers are zero when the request is made, the current upper bound is returned in A and the lower bounds in Q. To set the bounds, the request is made with the upper bounds in A and the lower bounds in Q. Both values must be in unprotected core and the upper value must be greater than the lower. Illegal values result in job termination. Each new request replaces the parameters from the previous request. At the beginning of a load, the entire unprotected area is made available again. The macro request format is:

CORE

The CORE request code is 11 and the calling sequence generated by the macro is:

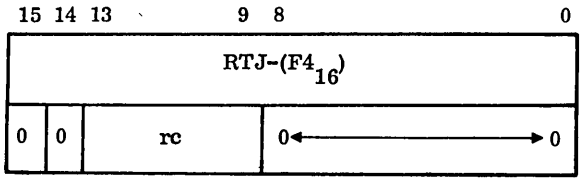


LOADER

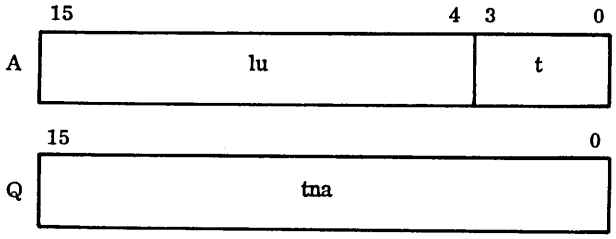
The LOADER request is available to unprotected programs at level 0 only. It is used to execute the mass-storage-resident relocatable binary loader. Parameters required must be in the A and Q registers at the time the request is made. The loader is placed in the uppermost part of unprotected core and loads programs into the area designated by a CORE request as available for loading. The macro request format is:

LOADER

The LOADER request code is 7 and the calling sequence generated by the macro is as follows:



Entry to the Loader Processor



Where: lu is the logical unit number of the input unit if a relocatable binary program is being loaded.

t is the type of loading operation. It can be used to execute the program when loading is completed. (See the following paragraphs for types of loading.)

tna is the entry point; the core address of the first of three sequential locations containing the entry point name. It is usually held elsewhere in the same program.

Example:

```
LDA = N$60      Load from logical unit number 6
ENQ   0
LOADER
```

Types of Loading

There are three types of loading, which are described in the sequence listed:

- Relocatable binary programs
- Subroutines
- Library programs

When relocatable binary programs are to be loaded, the t field of the A register contains 0; the lu field contains the logical unit number from which input is to be obtained.

If the leftmost bit of the lu field is 1, the standard binary input device is to be used and the loader refers to a location in the communications region containing the equipment ordinal. The Q register is ignored.

A relocatable program can be loaded from as many logical units as are required for execution on a single load operation. This is possible since no patch is performed by the loader until a subroutine loader function or the CREP directories link function is specified by the user program.

When the t field contains 1 and the lu field specifies the library unit, subroutines are loaded after relocatable binary programs. The Q register is ignored. The loader attempts to match external names with entry point names in its table. If successful, the appropriate routine from the program library is loaded for each match. If unsuccessful, the loader checks the program library directory and types the names of unpatched externals on the system print device. The operator must type:

*E To search the directory of core-resident part 0 and part 1 entry point names

*T To terminate the operation

* To continue loading with unpatched externals

When the t field equals 2, the loader enters the program directly into core from the program library and executes it immediately. The lu field is ignored. The Q register contains the address of the first of four sequential core locations in which the program name is stored in ASCII code, with the first character of the name appearing in the lower half of the first word. The program name is the entry point name and must appear in the program library directory.

Example:

```
          ENA    2
          LDQ  = XADRESS
          :
          :
ADDRESS   ALF    4, NAMEXX
```

When the t field equals 3, the loader produces a main memory map. This map includes the entry point table names and addresses. It may be produced subsequent to each subroutine load. The first word addresses of common and data storage reservation appear in the map as entry point addresses. The entry point names ***COM and ***DAT are used for common and data storage reservations. The lu field and the Q register are ignored.

When the t field contains a 4, the loader looks up the entry point name specified in A and Q as follows:

A register Bit 15 = 0 The name starts in the left character of the word

1 The name starts in the right character of the word

The locations containing the name should have the final character blank to terminate the name.

Q register Location of the name in ASCII

On exit, the A register contains the location of the name. If the name specified is not in the loader table, an E16 error message is typed and the loader waits for a legal name that is in the loader table.

When the t field contains a 5, the operation is equivalent to subroutine loading, but no memory map is produced.

When the t field contains a 6, the directories of the core-resident entry points (CREP and CREP1) can be searched first, instead of the program library directory. This function prints all unpatched externals and the operator can input an * or *T to continue or terminate.

When the t field contains a 7, the Q register specifies the core location to which subsequent data blocks should be relocated.

The following is a list of the loader calls:

t	Function	lu	tna
0	Load relocatable binary programs from any unit.	Input device	Ignored
1	Load from program library on library unit.	Ignored	Ignored
2	Load program from library unit and execute immediately.	Ignored	Location of Program name
3	Produce memory map.	Ignored	Ignored
4	Look up entry point name.	Ignored	Location of entry point name
5	Same as t = 1, but no memory map is printed.		
6	Search directory of core-resident entry points.	Ignored	Ignored
7	Initialize data base.	Ignored	Ignored

Termination of Loading

Program loading continues until interrupted by an EOL statement, an operating system control statement, or an error condition. The following exit parameters for the loader appear in the A register:

-0	Irrecoverable error
≠0	Load is terminated normally. The A register contains the transfer address or the entry point address of the look-up entry point function.

If there is insufficient core when the loader is active, the program load is paged to mass storage and the program read back into core by the loader request processor, after all linking operations have been completed. The transfer address is passed back to the loader as with a normal load operation.

If the bad operation was terminated by a job processor control command, that command is passed to the job processor and executed as the next control function.

GTFILE

The GTFILE request (13) is used to read permanent files stored on the program library. A permanent file is written in absolute binary format. This request can be used to load absolute programs from the program library.

The name of the file in ASCII must be provided separately in the program if the sector address is unknown. It may be up to six characters in length and it must be stored in a three-word (six-character) buffer. The program directory is searched for such a file name. If it is found, the file is read into the specified buffer. At this time, the address of the program on mass storage is saved in the last two words of the calling sequence parameter list. This allows subsequent uses of this source GTFILE call to read the file directly. Therefore, if the programmer knows the address of the file, the file name need not be specified at all. Instead, the file may be accessed directly, and not through the program library directory.

NOTE

The last two words of the calling sequence are not supplied by the macro. They are supplied by the caller, if known. If they are not specified by the caller, the sector address corresponding to the name is placed there when the request is processed. If the caller supplies an incorrect sector address, no diagnostic is given and the request results in bad data being read.

The macro format is:

GTFILE c,i,s,w_1,w_2,x,rp,cp,d

Where: c is the completion address; address of core location to which control transfers when an input/output operation is completed (refer to parameter x).

i is the pointer to the file name held elsewhere in the program. It has the form of a positive increment that is added to the address of the first word of the parameter list to form the address of the first word of a three-word block containing the ASCII name of the file. Locations before and after the request can be accessed because of memory wrap-around. If i is indirect, the parameter is the actual address of the first word of a three-word block containing the file name. For part 1, an indirect i is illegal. (Refer to the d parameter for additional information.)

s is the starting address of the main memory block into which the file, or portion of the file, is to be read (refer to parameters x and d).

w_1,w_2 are the first and last words of a partial file. These parameters must be blank if the entire file is to be used. The file words are numbered from 1 through n. If w_1 is nonzero and w_2 is zero, the remainder of the file starting at w_1 is read. If w_2 is beyond the end of the file, the job terminates. Since the file length is specified in sectors rather than words, w_2 must be specified if the file is desired in its exact length in words.

x is the relative/indirect indicator and modifies the meaning of c and s as indicated below.

(c) is indirect	This is illegal.
x is 0 or blank and c is direct	c is the completion address.
x ≠ 0 and is not blank and c is direct	c is a positive increment added to the address of the first word of the parameter list to form the completion address.
s is direct	x is meaningless and s is the starting address.
x is 0 or blank and (s) is indirect	s is the core location that contains the starting address of the block.

$x \neq 0$ or not blank and (s) is indirect	s is the core location that contains a positive increment added to the address of the first word of the parameter list to form the starting address of the block.

rp is the priority of the mass storage requests needed to complete a GTFILE request; it is always zero for unprotected requests.

cp is the priority of the completion address, the level at which the completion address is to be executed; it is always 1 for unprotected requests.

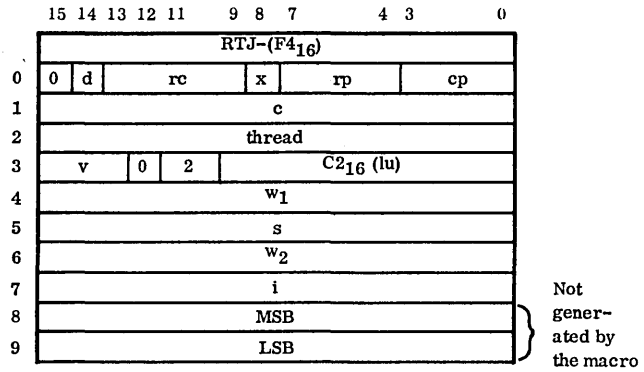
d is 0 Parameters operate as defined above.

1 x may not be set.

c and s are absolute 16-bit addresses.

i is a positive address increment which, when added to the address of the first word of the parameter list, gives the location of a three-word block containing the name of the desired file. This block must occur someplace after the request since backward relocation is impossible in this case.

The calling sequence generated by the macro is:



Where: rc is the request code.

thread is the thread location used to point to the next entry or threaded list.

v is the error code passed to the completion address.

2 is set automatically when the GTFILE macro call is used.

$C2_{16}$ is set automatically when the GTFILE macro call is used. That memory location holds the logical unit of the mass storage device containing the program directory.

MSB is the most significant bits of n. (n is starting sector of the file.)

LSB is the least significant bits of n.

When the GTFILE request is made with parameters w_1 and w_2 and sector LSB is specified as zero, the program library directory is searched to determine the location and length of the requested file. When determined, this length and sector LSB are placed in the parameter list. The user may save these parameters for future references to that file, thereby decreasing the time required to access the file. Conversely, if the same request is used to access a different file, w_1, w_2, and sector LSB must be appropriately set or zeroed.

STATUS

The STATUS request (3) is used to determine the status of an input/output device by accessing information from the physical device table for the specified logical unit. Refer to the note following the request description. The request macro format is:

STATUS lu,0,a,x,d

Where: lu is the logical unit; an ordinal in the logical equipment tables (refer to appendix C); modified by parameter a.

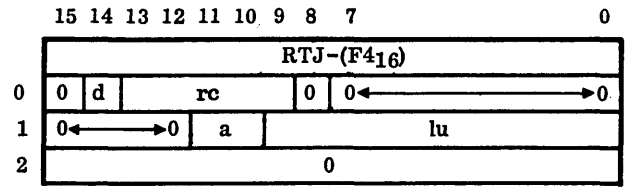
0 is the third word of the calling sequence; it must always be zero.

a is the absolute/indirect indicator, the same as the corresponding indicator for read/write requests.

x has no meaning; it must always be zero.

d is the part 1 request indicator (absolute parameter addresses).

The request code for STATUS is 3 and the calling sequence generated by the STATUS macro is as follows:



Following execution of the STATUS request, the A, Q, and I registers contain status information derived from the physical device table (PHYSTB) of the specified logical unit (see appendix C). The A register contains the hardware status reply (word 12 of the physical device table), the Q register contains word 8 of the PHYSTB (identifying the hardware type of the logical unit), and the I register contains the last core address used by the data transmission; i.e., the last word transferred to/from the peripheral device (word 11 of the physical device table minus 1).

NOTE

Since MSOS is a multiprogramming system, caution should be exercised in interpreting the results of the STATUS request. Since requests are executed on a priority basis, if more than one program is using a logical unit, it is difficult to determine which of the last operations created the status.

EXIT

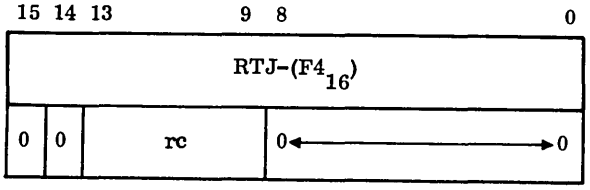
This request allows background programs to return control to the job processor. It signals completion of a job or of an interrupt routine. When computation is completed, the request notifies the operating system.

The interrupt stack is checked for exits to unprotected core. If any exist, the dispatcher is entered. If none exist, the request stacks are interrogated for requests originating in unprotected core. The job processor is entered for completion if no requests originated in unprotected core. It waits for completion if one or more requests originated in unprotected core.

A jump to entry point DISP from an unprotected program is treated as an EXIT request. The macro request format is:

```
EXIT
```

The EXIT request code is 5 and the calling sequence generated by the macro is as follows:



REQUEST RESTRICTIONS

Certain restrictions apply to the use of all requests executed from unprotected core. Violation of these restrictions results in job termination. If these restrictions are violated by requests from protected core, unpredictable results occur since limited error checking is performed.

- Invalid addresses – Addresses must be valid for the requesting program. A program in unprotected core cannot have interrupt or control information addresses in protected core. An example of a control information address is the address of an area of core from or to which a block is to be transferred.

- Illegal logical unit – The logical unit number must be legal. Logical unit numbers of zero or greater than the largest available logical unit number defined in LOG1A table are illegal.

- Illegal control information – Requests must not contain illegal control information (any information that can cause destruction of part of the system). For example, a READ into protected core or a WRITE into system areas of mass storage is illegal.

- Busy requests – All input/output requests are threaded using the third word of the parameter list. A given input/output request cannot be repeated until it is taken off the thread (completed). An attempt to repeat a busy request in protected core is not processed. Instead, control returns to the caller at the normal place and the Q register is set negative to avoid delays at high priority levels. An attempt to repeat a busy request by an unprotected program is automatically repeated until its thread is cleared. A limit of five requests may be queued from unprotected core.

SWAPPING CORE

Swapping is a process in which the contents of unprotected core are stored on mass storage to make more protected core available for assignment by SPACE requests. The scratch area of mass storage normally used by jobs is unchanged.

If unprotected core is in part 0, the unprotected area is protected and combined with allocatable core. If unprotected core is in part 1, the unprotected area partition is protected and made available as a nonbusy partition for protected programs scheduled for that partition. The unprotected partition has its own special thread word and its boundaries are defined by $F6_{16}$ and $F7_{16}$.

A core swap is automatically performed at system start-up.

Swapping and subsequent swapping are attempted under the following conditions:

- A SPACE request cannot be satisfied within the protected area set aside for this purpose.

- Input/output from the unprotected area is not active. If the buffered version (BPROTK) of the protect processor is used, a swap may occur while unprotected I/O is in progress if the size of the request does not exceed 96 words and a request is not currently buffered.

- A minimum time (SYSDAT program parameter) has elapsed since the last swap.

- The request's completion priority is greater than two.
- The part 0 swap inhibit flag is not set if unprotected core is in part 0.

When all the above conditions exist, the contents of unprotected core are written in a reserved area of mass storage and unprotected core is set up as protected. A null loop is started at priority level 2 to prevent operation of any portion of a job. The newly protected area is made available for assignment by the SPACE program.

After swapping has occurred, subsequent RELEAS requests cause examination of the swapped area in core to ensure that it has been totally released. If it has, the area previously protected by swapping is set unprotected and information formerly in that area is returned from mass storage. The priority level 2 loop is discontinued and the newly unprotected area becomes available for SPACE assignment. Job processing can then resume at the point of interruption as if swapping had not occurred.

When unprotected core is configured within part 0, the protected core area used for space assignment must be continguous with unprotected core if swapping is to work properly. Therefore, the space request processor module must be the last part 0 core-resident program presented during system initialization.

Upon termination of job processing, a swap is automatically requested. The swap remains in effect until the operator again requests job processing. This allows the core allocator to allocate space in the entire area, including the area used by unprotected programs, without the delays caused by unnecessary swapping. Only programs of level 3 or above can run when job processing is not in progress, because of the level 2 loop started by the swap.

STANDARD SYSTEM INPUT/ OUTPUT DEVICES

The logical units of the devices listed are stored into the stated core locations. If these locations are used in system requests, changing equipment does not require reassembly.

Device	Core Location	FORTRAN Logical Unit
Standard input device †	$F9_{16}$	1
Standard binary output device †	FA_{16}	2
Standard print output device †	FB_{16}	3
Output comment device	FC_{16}	4
Input comment device	FD_{16}	
Mass storage scratch	$B3_{16}$	
Mass storage library	$C2_{16}$	

† The operator can change these values by an *K statement from the comment device (section 9). All programs can, therefore, address these particular units (indirectly) or determine their numbers by interrogating the communications region.

MSOS drivers support operation of peripheral devices. They also simulate peripheral devices via software to provide additional system capability. Each peripheral device in the computer system is associated with a device driver, which is the only software allowed to give direct commands to the device.

The driver controls execution of READ, FREAD, WRITE, FWRITE, and MOTION requests.

READ/WRITE calls request processors to transfer data between the specified input/output device and memory. The word count specified in the request determines the end of the transfer.

Format read (FREAD) and write (FWRITE) requests cause records to be read or written in a specific format. A particular format is associated with each device.

Further explanation and parameter descriptions of READ/FREAD/WRITE/FWRITE are in section 3.

MOTION requests are handled by each driver. The meaning and function of each MOTION command may differ for each device. Refer to section 3 for parameter descriptions and device capabilities.

A complete description of the drivers is found in the MSOS Peripheral Drivers Manual.

The CYBER 18/1700 File Manager is a general-purpose file management package consisting of a request supervisor and a collection of request processors. The supervisor resides in core and the request processors on mass storage; core requirements are minimized by bringing in individual request processors only as they are needed.

Communication between the user's program and the file manager is provided by predefined system macros in MSOS assembler language and by calls to predefined system subroutines in MS FORTRAN. These macros and subroutine calls enable the user to request any of the services provided by the file manager. Provision is made so that the user can specify any needed options and pass any required information as parameters.

The file manager creates and maintains either sequential or indexed files. The records in any file may be variable in length and may be added, replaced, or removed at any time after the file has been defined and before it is released.

A sequential file is one in which each new record is added immediately following the last record stored in the file. These records must be retrieved in the same sequence in which they were stored and cannot be retrieved at random. Thus, they are retrieved on a FIFO basis.

An indexed file is one in which each record has an identifier or key (surname, social security number, etc.). These records are stored sequentially with a key value; records with the same key value may be linked together. These records may be retrieved sequentially or a specific record may be retrieved by using its key value. Consult the File Manager Reference Manual for complete information.

To minimize mass memory I/O traffic, the File Manager is designed to allow file information to remain in allocatable core until a time-out occurs, at which time the information is updated on mass storage. Users are cautioned that abnormal system stops and autoloads can destroy this information and eventually cause fatal file errors.

If the system contains a file manager, a file validity check is performed each time the system is autoloaded. The check is preceded by the message:

CHECKING FILES -

on the system comment device. The check consists of a trace of all file space threads on mass storage. If the threads are found to be valid, an OK is printed. If errors are found, the user is given the option to continue with the autoload or to purge all the system files (i.e., all pointers to the file manager space pool are reset to a state that indicates that no files are defined). If the second option is selected, the files have to be reloaded from a user-written back-up dump.

STORAGE AND RETRIEVAL

The file manager stores and retrieves information in three basic ways:

- Sequential
- Indexed
- Direct

In addition, variations for storage and retrieval are provided by combinations of the preceding and special options. The variations are:

- Indexed-ordered
- Indexed-linked

SEQUENTIAL

Records are stored one at a time immediately following the last record stored and retrieved one at a time in the same order they were stored starting from the beginning of the file.

Sequential access is best suited for retrieving all records on a FIFO basis. It is not suited to retrieving a particular record since all preceding records must first be retrieved.

NOTE

All files may be accessed sequentially.

INDEXED

Indexed access is best suited for access of a specific record. Each record may be indexed by only one key. The key may be one or more words in length. Since all files are sequential, an indexed file may be termed indexed-sequential. Indexed access is only possible from an indexed file.

A particular record can be stored and retrieved via a key. Each record key value can be translated into an index which can provide relatively quick access to the record. Indexed files require extra file space for the keys and key directories.

DIRECT

Direct access is best suited for frequently accessed records or records which the user desires to link together (e.g., in some types of list structure) using pointers within the records themselves.

Direct procedures are normally used in updating records and in forming list structures. Since the File Manager provides record pointers for all records, all the files may be accessed directly.

VARIATIONS

Indexed-Ordered

When the indexed-ordered option is selected, indexed records can be retrieved in a manner similar to sequential retrieval. However, instead of on a FIFO basis, records are retrieved starting at the record with the lowest numeric key value (or the key value specified in the first of the repeated indexed-ordered retrieve) and continuing through to the record with the highest numeric key value. When this type of access is used, a sort of the key values is done; therefore, the key value must be one word in length. It is recommended that key values for an indexed-ordered file include only non-negative values.

Indexed-Linked

In an indexed file, each record normally has a unique key value. However, if the indexed-linked option is selected, records with the same key value are linked together in either a LIFO or FIFO manner. The records are linked by allocating two words of each record for the linking record pointer. The retrieval of these records is an example of a list structure. LIFO or FIFO linking is specified by the user when a file is defined as indexed.

List Structures

Records may be retrieved as though they were part of a list structure by using the record pointers supplied by the file manager and the direct method of retrieval. The user may form complex list structures by linking forward, backward, ring, sublist, etc. A record may be a member of an indefinite number of lists as long as two words for a record pointer are reserved in the record for each list. An example of a list structure is the indexed-linked file.

FILE MANAGER GENERAL DESCRIPTION

FILE REQUESTS

Four types of file requests are described in File Request Descriptions and Calls. They are:

- Specification — Specification requests provide for:
  - -Defining a file
  - -Defining an indexed file
  - -Locking a file
  - -Unlocking a file
  - -Releasing a file
- Sequential
- Indexed
- Direct

  These three file requests are used to store and retrieve information.

The file manager executes a request at the caller's priority level. If, however, the file manager is executing a previous request, the request is queued by its priority level and is not executed until the currently active request and any higher level waiting requests have been executed.

Associated with each request is a 12-word temporary buffer and one indicator word giving status information. The buffer is used to process the file request; the indicator word denotes the status of the file request upon completion. Each bit of the indicator word which is a nonzero signifies an abnormal occurrence.

RECORD FORMAT

Each variable-length record is composed of three sections: header word, record pointers, and data words.

Header Word

The first word of each record is reserved exclusively for the header word. The file manager sets this word to the total length of the record when this record is stored. Once a record is defined, its length (and consequently the header word) cannot be changed.

NOTE

When storing and retrieving a record, the number of words in a record must include the header word.

Record Pointers

A record in an indexed-linked file includes a record pointer. A record pointer is a two-word mass-storage address which points to another record on mass storage. The first word is the sector location of the file record block in which this record resides. The second word contains the word the record starts in. If a file is indexed-linked, the second and third words are reserved for the record pointer, which points to the last record that was stored with the same key value. This is the same format as the recptr parameter passed back to the user from the STOSEQ and STOIDX requests.

Data Words

Each record may have zero or more data words which contain the actual record information. The information may be binary or ASCII.

UPDATE PROTECTION

Whenever a record is to be updated, the user must retrieve the record and lock the file with a unique file combination, subsequently storing the updated record and unlocking the file with the same file combination, utilizing the store direct request. More than one record may be retrieved, updated, and restored as long as the same file combination that was used to lock the file is supplied. Note that the file should not be locked for an extended period of time because other users, who may also wish to update, cannot access the file until it is unlocked. Thus, a retrieve, which attempts to lock an already locked file with a different combination, is queued and cannot be executed until the file is unlocked.

If a number of files are to be locked, it is advisable to lock and unlock the files in a given sequence. For example, lock files in ascending numerical order and unlock them in descending numerical order.

A retrieve without a file combination or a store of a new record is permitted on a locked file with the understanding that one or more records of that file are in the process of being updated. Note that an update into an unlocked file or a locked file using an incorrect file combination results in a file request error.

The file combination must be unique so that no two requests use the same file combination. This can be accomplished by using the ASSIGN statement in FORTRAN or the RTJ instruction in assembly language.

UNPROTECTED FILE REQUESTS

Unprotected programs are assumed not to be error-free; therefore, certain restrictions have been placed on unprotected file requests.

An unprotected file request cannot update a record in a file because it cannot use the store direct request. This restriction is imposed because the file manager has no way to check the validity of the record pointer in the store direct request. The restriction is mitigated by the assumption that background programs primarily retrieve records (for example, data reduction, analysis, etc.) and that records can always be retrieved, updated, and stored as new records in another unprotected file.

Since updates cannot be done, file locking is illegal for unprotected file requests. Note that unprotected file requests may not store records into or remove records from files that were defined by protected programs.

NOTE

If there is not enough allocatable core for both the file manager and job processor modules, file requests from background can hang batch processing indefinitely.

REQUIREMENTS AND LIMITATIONS

The file manager requires certain information to establish the file structure and imposes limitations on those files.

Maximum Record Length

The effective maximum record length and file record block length are determined as a function of the maximum record length specified by the define file request for each file. It places a maximum limit on the length of the records for that file, and also establishes a block of sector(s) that will be allocated when the first record is stored into the file. Subsequent records are stored into this block until it is full, then another equal block of sector(s) is automatically allocated. This process is continued as long as there is mass memory space available. Thus, a file record block may contain one or more records.

The effective maximum record length is equal to the specified maximum record length if the specified maximum record length plus 3 is equal to an integral multiple of 96. Otherwise, the effective maximum record length is equal to the least integer value n such that n is greater than the specified maximum record length, and $n = 96 \cdot m - 3$ for some positive integer m. Thus, specified maximum record length values of 3, 93, and 94 would result in effective maximum record lengths of 93, 93, and 189, respectively.

Expected Number of Records With Different Key Values

The expected number of records with different key values is specified by the define file indexed request number for each indexed file. Note that if a file is not indexed-linked, this is equivalent to the number of records in the file. The expected number of records with different key values establishes the structure of the indexed directories. A relatively accurate estimate is important if the number of expected key values exceeds 8,464.

Too low an estimate may result in more mass storage accesses per indexed request, while too high an estimate may result in excessive core allocation for the indexed directories per indexed request.

Parameter Limitations

The following limitations are necessary:

File number range	1 through 32,767
Record length range	1 through 32,767
Number of expected records range (with different key values)	1 through 32,767
Key value length range	1 through 63
File combination range	1 through 32,767
Key value range for indexed-ordered files	0 through 32,767

CAUTION

Users are warned that programs making file manager requests that contain relative parameters do not execute properly in partitioned core or at addresses above 8000_{16}.

FILE REQUEST DESCRIPTIONS AND CALLS

All file request calls to the file manager may be written as FORTRAN-type calls or as assembly language macros as in the following descriptions. A comprehensive set of macros is provided in the macro library for all file manager functions. These macros minimize the task of the assembly language programmer of making file manager calls.

Specification descriptions and calls are:

Type	Call
Define file (numbered files)	DEFFIL
Define file indexed (keyword files; these also have a file number)	DEFIDX
Lock file	LOKFIL
Unlock file	UNLFIL
Release file	RELFIL

Sequential record request descriptions and calls are:

Type	Call
Store records sequentially	STOSEQ
Retrieve sequential records	RTVSEQ

Indexed (keyword) record request descriptions and calls are:

Type	Call
Store indexed records	STOIDX
Retrieve indexed records	RTVIDX
Retrieve indexed-ordered records	RTVIDO

Direct record request descriptions and calls are:

Type	Call
Store records directly	STODIR
Retrieve records directly	RTVDIR

The System Initializer is a group of programs used to construct a Mass Storage Operating System (MSOS) from an installation file. The initializer constructs core memory and mass memory areas based on the information in the installation file and produces a memory map of these areas.

The basic elements of the system initializer are:

- The control statement handler
- Device drivers
- The program loader
- Pre-initialization set-up
- Post-initialization set-up

CONTROL STATEMENT HANDLER

The system initializer control statements are processed by the Control Statement Handler (CONTRL). The following control statements are used for system initialization:

Code	Function
*C	Define memory map list unit.
*D	Assign labeled COMMON base address.
*G	Write address tags on disk unit.
*H	Run surface test on disk unit.
*I	Define input unit.
*L	Load part 0 core resident.
*LP	Load part 1 core resident.
*M	Load system library in allocatable core.
*MP	Load system library in partitioned core.
*O	Define mass memory unit.
*S	Patch external values.
*T	Terminate initialization.
*U	Read control statements from comment unit.
*V	Read control statements from input unit.
*Y	Core-resident ordinal specification
*YM	Mass-memory-resident ordinal specification
*	Dummy character – used for comment records

DEVICE SPECIFICATION

*C Statement

This statement assigns the standard list output to the logical unit specified by lu (6, 7, or 8). The equipment code, e, of the device is optional. If e is specified, this value (four hexadecimal digits) must correspond to the WES word for the device status. The initial set-up is for logical unit 6 and equipment code 0091 (equipment 4). The logical unit assignments are given in the Hardware Device Drivers section. The *C format is:

```
*C,lu,e
```

*I Statement

This statement assigns the standard input to the logical unit specified by lu (1, 2, or 3). The equipment code, e, of the device is optional. If e is specified, this value (four hexadecimal digits) must correspond to the WES word for the device status. The initial set-up is for logical unit 3 and equipment code 0381 (equipment 7 for 1700 Systems), or equipment code 0480 (equipment 9 for CYBER 18 Systems). The *I format is:

```
*I,lu,e
```

*O Statement

This statement assigns the standard mass memory device to the logical unit specified by lu (4). The equipment code, e, of the device is optional. If e is specified, this value (four hexadecimal digits) must correspond to the WES word for the device status. The initial set-up is for logical unit 4 and equipment code 0181 (equipment 3 for 1700 Systems), or equipment code 0700 (equipment 14 for CYBER 18 Systems). The *O format is:

```
*O,lu,e
```

*U Statement

This statement causes the system initializer to interrogate the comment unit for control statements. The initializer begins operation in this mode. The comment unit is always unit 6, the teletypewriter or display console. The *U format is:

```
*U
```

*V Statement

This statement causes the system initializer to interrogate the input unit for control statements. The *V format is:

*V

DISK TESTING

*G Statement

This statement causes the system initializer to write address tags on logical unit 4, the disk. On CYBER 18 Systems, *G causes the system initializer to write address tags and data on the entire disk. The *G format is:

*G

*H Statement

This statement causes the system initializer to run a disk surface test to identify errors. The test is run on logical unit 4. hhhh is the maximum sector address for the surface test (sectors 0 through hhhh). The *H format is:

*H,hhhh

Performance of this function requires several hours.

SYSTEM DIRECTORY ORGANIZATION

*Y Statement

This statement sets up the core-resident entries in the system directory. The directory names aaaaaa,bbbbbb,... are placed in the CREP table as entry points. The decimal ordinal numbers, A,B,... correspond to the program(s) loaded under the Ath, Bth, etc. *L and *LP core-resident load control specifications. *Y statements must precede *YM statements. The *Y format is:

*Y,aaaaaa,A,bbbbbb,B,...

*YM Statement

This statement sets up the mass-memory-resident entries in the system directory. The directory names cccccc,dddddd,... are placed in the CREP table as entry points. The decimal ordinal numbers, C,D,... correspond to the program(s) loaded under the Cth, Dth, etc. *M and *MP mass-memory-resident load control specificaitons. The release version allows a maximum of 256 ordinals. The *YM format is:

*YM,cccccc,C,dddddd,D,...

EXTERNAL STRING PATCHING

*S Statement Program

The statement *S,n,hhhh assigns the hexadecimal value hhhh to the entry point name n and places both in the CREP table. Previously defined external strings are patched with hhhh. If an *S statement occurs within an *M or *MP load, it is not put into the CREP table and is effective only for that particular load sequence. It patches to core-resident programs as required.

The statement *S,n,S assigns the value of the current mass-storage sector to entry point name n. It permits dynamic assignment of values to symbolic names and places the name/sector value in the CREP table.

The statement *S,n,P assigns the current value of the program base to entry point name n. If an *S is encountered during an *L or *LP load, n is set equal to the current absolute program counter. If an *S is encountered during an *M or *MP load, n is set equal to the current relative program counter.

PROGRAM LOADING

Control statement sequences have the following order (with the exception of the *D statement):

*L,*LP,*M,*MP,*T

*D Statement

This statement resets the start of labeled COMMON to the current program base. The new labeled COMMON definition must follow the *D entry. The *D statements must follow the *L statement and precede the *M statement. The *D format is:

*D

*L Statement

This statement loads core-resident part 0 programs. The hhhh parameter is the hexadecimal location where the program resides in the system when initialization is completed. Any space between hhhh and the top of previously stored programs becomes unavailable for subsequent storage, and the program starting at hhhh cannot extend past the end of part 0. If hhhh is not present, the initializer loads relocatable programs from the input device into unused core locations beginning as close to location 0000 as possible. The *L format is:

*L,hhhh

*LP Statement

This statement loads core-resident part 1 programs. The hhhh parameter defines the hexadecimal location of the program in the system. Any space between hhhh and the top of previously stored programs becomes unavailable for subsequent storage. The program starting at hhhh must exceed the end of part 0, but cannot extend beyond the limits of part 1. if hhhh is not used, the program is loaded at the next available core location. The *LP format is:

*LP,hhhh

Part 1 data (labeled COMMON) blocks can be linked to the last block in part 0. Note the restrictions for linking *M and *MP data blocks.

*M Statement

This statement loads and absolutizes mass-memory-resident programs to run in part 0 (allocatable core). The *M format is:

*M,hhhh,s

Where: hhhh is the base address used to absolutize a program to a previously defined entry point name plus the increment to be added to the entry to form the sector address of the program on mass storage. Loading is relative to hhhh; if hhhh is omitted, loading is relative to zero.

s is the sector address in mass storage of the program or block of programs to be stored. If s is omitted, the next available sector is assumed.

The initializer loads programs in relocatable form from the input device and places them in absolute form on mass storage.

CAUTION

The maximum sector available to the initializer loader is 1540_{16}. The area preceding this sector is used as intermediate storage for various tables. Depending upon the number of entry points, etc., the size of this area should nominally be considered as 300_{16} sectors. Loading of executable code into the area between sectors 1240_{16} and 1540_{16} by *M or *MP statements should be avoided.

Programs loaded under a single *M statement are linked to one another, but not to programs loaded as a result of other *M statements. They may also link to programs loaded under either *L or *LP statements. Entry points declared under an *M load may not be used for other *M loads as they are not retained in the core-resident entry points tables.

All programs loaded under *M must be written as run-anywhere. The total length on any program being loaded under an *M statement must not exceed the total length from the start of the load (usually 0000_{16} on *M loads) to $FFFF_{16}$.

The mass-storage address that follows the last address containing information about the program block becomes the first address available for subsequent loading.

Any mass-storage address between s and the last address used is unavailable for subsequent storage.

Programs may not be stored on mass storage in areas containing previously stored programs.

COMMON linkage is to system block and system labeled COMMON and cannot be assigned within the ordinal. If no system COMMON has been defined, an *M generates system COMMON.

An *M load that declares data must have previously loaded the part 0 data block. Linkage is always to the last data block loaded in the appropriate part.

Data blocks cannot be present within an *M or *MP load. Any data that is to be defined within these data blocks must be preset by the *L or *LP data block.

*MP Statement

This statement is used to load and absolutize mass-memory-resident programs to run in part 1 (partitioned core). The *MP format is:

*MP,pp,nn,ssss

Where: pp is the number of the starting partition to which the program is absolutized (0 through 15). If pp is omitted, an error message occurs.

nn is the number of partitions required by the program (1 through 16). If nn is omitted, an error message occurs.

ssss is the mass-storage address at which the program is to be stored. If ssss is omitted, the next available sector is assumed.

The system initializer loads programs in relocatable format from the input device and places them in absolute form on mass storage. Loading is relative to the partition address. The programs are linked to core-resident programs loaded by *L and *LP statements.

Programs loaded under a single *MP statement are linked to one another, but not to programs loaded as a result of other *MP statements. Entry points declared under an *MP load may not be used for other *MP loads as they are not retained in the CREP1 table.

The total length of any programs being loaded under an *M statement must not exceed the total length from the starting partition address as defined in PARTBL and $FFFF_{16}$.

Any mass-storage addresses between ssss and the last address used are unavailable for subsequent storage.

Programs may not be stored on mass storage in areas containing previously stored programs. Therefore, ssss must be greater than the last sector used.

COMMON linkage is to system blank and system labeled COMMON. Labeled COMMON cannot be assigned within the partition. If no system COMMON has been defined, an *MP generates COMMON within the partition.

An *MP load that declares data must have previously loaded the part 1 data block. Linkage is always to the last data block loaded in the appropriate part.

Data blocks cannot be preset within an *M or *MP load. Any data that is to be defined within these data blocks must be preset by the *L or *LP data block.

*T Statement

This statement signifies the end of the load and causes the initializer to perform postload initialization and to terminate. The *T format is:

```
*T
```

At the completion of initialization the entry point DATBAS and the extended communications region word 17 (appendix B) contains the address of the last data block loaded. Word 25 contains the last address of the last data block loaded.

COMMENT CONTROL

* Statement

This statement may be used as a listing separator for comments, etc., when followed by a blank. The * is a null statement and must be followed by another control statement (e.g., *L, *M). The * format is:

```
*
```

HARDWARE DEVICE DRIVERS

The System Initializer provides device drivers for many of the peripherals supported under MSOS. One driver of any device type may be present in a given System Initializer. The drivers in the System Initializer do not use interrupts but operate the devices on the basis of the device status condition. In this mode of operation, device operation and timing may differ from the interrupt-driven MSOS driver. The logical units used by the initializer are predefined in the range 1 through 8. The minimum initializer requirement is an input driver, comment driver, and mass-memory driver. Dummy routines are provided to satisfy external linkages for missing routines. Table 6-1 defines the logical units, driver names, devices supported, and dummy names.

DRIVER OPERATION

Input Drivers

The input drivers are controlled by the input driver interface, IDRIV. IDRIV calls the device driver by a return jump instruction which passes the following to the driver:

- A register – The buffer first word address for read
- Q register – The number of words to be read

The driver returns to IDRIV with the error indicator (0 for error and 1 for no error) in the A register. If an error occurred, the following error information is passed:

- Q register – The error code
- I register – The last hardware status

Mass-Memory Drivers

The mass-memory drivers are controlled by the mass-memory driver interface, MDRIV. MDRIV calls the device driver by a return jump instruction which passes the following to the driver:

- A register – If set positive, buffer first word address
  If set negative zero, write address tags (disk only)
  On CYBER 18, writes address tags and data.
- Q register – If set positive, number of words to be read
  If set negative, complement of the number of words to write
- I register – The starting sector address

The driver exits with the error indicator (0 for error and -0 for no error) in the A register. If an error occurred, the following error information is passed:

- Q register – The error code
- I register – The last hardware status

TABLE 6-1. HARDWARE DEVICE DRIVERS

Logical Unit	Driver Name	Hardware Devices	Equipment Code/WES	Dummy Name†	Other Requirements
1	QPTAPE	1721/1722 1777	1/00A1	QPTDMY	
	QPTAP1	1720-1			
2	QCARD	1726/405 1726/1706/405 1728/430 1729-2 1729-3 1829-30/60	11/0581 11/1581 11/05A1 11/0581 11/0581 11/0581	QCDDMY	ASCII63 conversion requires CR026. ASCII68 conversion requires CR029.
3	QMT9TK	1732-1/609 1732-1/1706/609 1732-2/615-93	7/0381 7/1381 7/0381	QMTDMY	Unit 0 used
3	QMT7TK	1731/601 1731/1706/601 1732-1/608 1732-1/1706/608 1732-2/615-73	7/0381 7/1381 7/0381 7/1381 7/0381	QMTDMY	Unit 0 used
3	QMLCT9	1860-92	9/0480	QMTDMY	Unit 0 used
3	QMLCT7	1860-72	9/0480	QMTDMY	Unit 0 used
3	QM7LS9	1860-5/6	7/0601	QMTDMY	Unit 0 used
4	QDK85X	1733-1/853/854 1738/853/854	3/0181		Unit 0 used
4	Q17391	1739-1	3/0181		
4	Q17332	1732-2/856-2/856-4	3/0181		Unit 0 used
4	Q1751	1751	2/0101		
4	Q1751	1752	2/0101		
4	Q18331	1833-1	14/0700		Unit 0 used
4	Q18334	1833-4	14/0700		Unit 0 used
6††	Q1711	1711 1712 1713 713-10	1/0091		
6	Q1810	1811	1/0091		
7	Q40421	1740/501 1742-1	4/0201	QPRDMY	
7	Q42312	1742-30 1742-120	4/0201	QPRDMY	1742-120 requires train image T5954
7	Q18277	1827-5/7			1843-2 communication line adapter
8	DUMMY†††				

† Used to allow linkage of unused driver entry points

†† Logical unit 5 is reserved for future use.

††† Used when no device action is desired

DRIVER ERRORS

The IDRIV module of the initializer reports device failures on the initializer comment device as:

```
L,xx FAILED yy (zzzz)
ACTION
```

Where: xx is the failed logical unit.

yy is the error code (same as for MSOS drivers).

zzzz is the last hardware status.

The response to the error takes one of two forms:

- RP – Repeat the operation.
- CU – Abort the operation and return to the comment unit for a subsequent control statement.

PRELOAD INITIALIZATION

The preload module of the initializer, I1, creates the system directory, intializes the loader table and, in general, prepares for the program loading operations. This module is operated when the first *L statement is encountered.

LOADER

The loader module of the System Initializer, ILOAD, handles the loading and linking of relocatable binary programs. The binary block handling is identical to the MSOS loader. Refer to section 12 for details.

The loader data block is resident in the LDRTBL module.

The loader is a paging type that uses mass memory for program storage, entry/external linkage, and intermediate storage. The only system components that are retained in core during the initialization operation are SYSDAT and the system directory. All core-resident linkage is done following the first *M statement (a pause occurs in the load operation). The loader creates the core image, CREP and CREP1 tables.

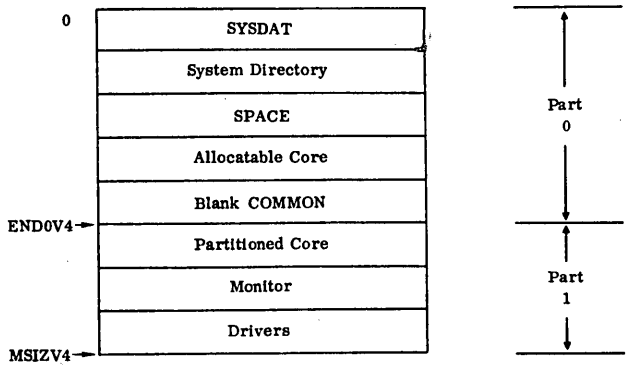
POSTLOAD INITIALIZATION

The postload module of the initializer, I2, creates the autoload program by placing system load parameters into the mass-memory device driver of the system. The mass-memory initializer driver becomes the autload program at this point.

SYSTEM MEMORY MAPS

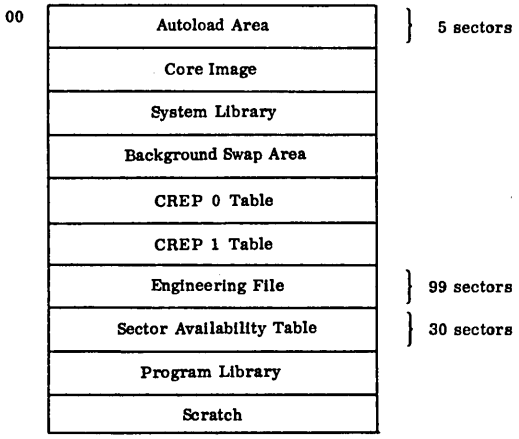
The initializer produces a memory map for the programs loaded as a function of the initialization process (unless the dummy unit is used for list output).

The following is the standard core-resident system layout:



The system may be reconfigured in several other forms. Consult the MSOS Customization Manual for further information.

The following is the standard mass-memory system layout:



ERROR RECOVERY

The initializer handles error recovery and flags error conditions as they occur. Most error conditions are immediately recoverable, but if an irrecoverable loading error occurs in the loading of a program, the initializer bypasses the remainder of the program and continues loading the next program. ERROR 17 appears on the comment device.

Table 6-2 identifies some of the problems that may cause initializer malfunctions.

TABLE 6-2. PROBLEMS CAUSING INITIALIZER MALFUNCTIONS

Problem	Cause
Initializer stops while loading the SYSDAT program	Index I (location FF_{16}) is not assembled in SYSDAT as a BSS(1). Locations 0 and FF_{16} usually contain the same value, which is the address of the initializer's constant table. The first *L control statement tried to load SYSDAT into the system library (an *Y,PROG,1 statement has been used). The SYSDAT program establishes the location of the system directory and therefore cannot be placed in the directory. This can be avoided by changing the first *Y statement to *Y,PROG,2.
Initializer stops or restarts during loading	Data has been stored over the initializer or a previously loaded program link string by an ORG instruction. Locate the ORG instruction.
Initializer terminates input or output	One of the following: • The requested device is not turned on. • The requested device is not ready and is locally cleared. • The equipment or station is not properly prepared for the initializer. • A hardware malfunction exists.
Initializer skips the next program after an *V statement	When the *V statement instructs the initializer to read subsequent control statements from the binary input device, the record read may be the NAM block of the program which cannot be recognized as a control statement. Either place a control statement at the input device before typing *V or type * instead of *V.
Job processor functions partially	When certain functions of the Job Processor are not working, it may be a system problem, or the construction of the system library may not correspond to the order in the *Y and *YM statements.
No autoload after successful initialization	The cause may be an improperly constructed interrupt trap or priority structure or a missing driver.

The Small Computer Maintenance Monitor (SCMM) provides a method of on-line hardware error detection for 1700 Computer Systems. SCMM consists of a main program and one test program for each I/O device to be tested. The main program is loaded into the operating system as a system ordinal and the tests are placed in the program library. SCMM runs at the lowest foreground priority, and all programs are run-anywhere. This section is intended as a general description only. For detailed instructions, refer to the 1700 Systems Small Computer Maintenance Monitor Reference Manual. SCMM is not applicable on CYBER 18-20 or 18-30 Timeshare Computer Systems.

ENTRY TO SCMM

Operator Entry SCMM Reply	Comments
Press manual interrupt	
MI	
SCMM	Requests SCMM
SCMM IN	
MM/DD/YY HHMM	
CONTROL, TEST ID	

Subsequent calls to the executive are made in the same way except that the SCMM response is CONTROL, TEST ID.

The operator responds by typing in the following control and test identification mnemonics:

ccc,ttt (form 1)

or

ccc (form 2)

Where: ccc is any SCMM control word:

SRT – Start		
STP – Stop		
PRM – Parameters	Use form 1	
NPT – Print inhibit		
PRT – Print re-enable		
LST – List		
CLR – Clear	Use form 2	
XIT – Exit		

ttt is any SCMM test that is loaded into the program library. (The operator can obtain a list of existing programs by use of the SCMM control word LST.)

If the operator's request was SRT,xxx (start a test) or PRM,xxx (return to parameter entry section of test), some of the following questions may be asked by the test in question.

DLU,LU	Diagnostic logical unit or logical unit number (decimal)
SECTIONS	Sections of the test; to request that a section be set into execution, construct a hexadecimal word so that the bit corresponding to the desired section number is set. For example, if tests 2 and 4 were desired, 0014 would be entered in this space. Bits for which no test section exists are ignored, so to obtain all sections of a test, the operator may enter FFFF. Test sections are listed in table 7-1.
RUNS LINES CARDS RECORDS	Number of runs, lines, cards, records, etc. desired in test; any decimal number up to 8000 may be entered. If 8000 is entered, the test executes indefinitely until the operator stops the test using conversational mode.
DENSITY	Decimal tape density (200, 556, 800, or 1600). See example in figure 7-1 of multiple test requesting magnetic tape test.
BEG SEC END SEC BEG TRK END TRK	Beginning/ending sector or track address of the disk or drum under test. The entry is hexadecimal.
WEMS	Equipment, module, and slot of IOM device being tested
BEG ADR, END ADR	1501 and 1536 tests; channel (points) numbers
ADC BIT TYPE	1501 and 1536 tests; analog-to-digital converter bit, type 12 or 14
SCALE	1501 and 1536 tests; scales histogram to value controls
READINGS PER RUN	1501 and 1536 tests; number of times point is to be read per run
% FS COUNTS/ 100% FS COUNTS	1501 and 1536 tests; expected input for a 0 percent full scale and 100 percent full scale
% FS, GAIN	1536 test; expected percent of full scale; gain 1, 10, 100, 1000 that applied to the analog-to-digital converter
% FS	1501 test; expected percent of full scale

INTERRUPT LINE — 1572-1 test; interrupt line of timer (1 to 15)

COMPUTER TYPE — 1572-1 test; computer type:

0	1784-2
1	1784-1
2	1704-1714
3	1774 SC

LST INT CNT — Number of LST interrupts that are used in the 1572-1 test

MULT — Value that is to be loaded into the 1572-1 multiplier register

SYNC — Sync enable word:

Bit 1	=	Sync 1 LST
Bit 2	=	Sync 2 SRG
Bit 15	=	Disable type out of results

OUT LU — MSOS logical unit for the 1555 or 1553

IN LU — MSOS logical unit for the 1544

SWITCH — Used to set up test for open- or closed-type contacts on 1555 and 1553 tests

When all necessary parameters have been correctly entered, conversational mode is terminated by the program. The operator must re-initiate conversational mode by typing manual interrupt and SCMM for each control word to be entered. At any time during the entry of the test parameters, the operator can abort the test by typing a question mark (?) in the position of a parameter.

TABLE 7-1. SCMM TEST SECTIONS

Test	Section	Description
TTY† (Teletypewriter)	1	Legal characters (sliding)
	2	Echo of operator input line (once)
	3	Echo of operator input line (repetitive)
PRT (Line Printer)	1	Variable length buffer test
	2	Ripple pattern test
	3	Full line of same character
	4	Alternate even and odd hammer test
	5	Single and double line spacing
	6	Six and eight lines per inch (1742 only)
MTT (Magnetic Tape)	1	Worst case pattern
	2	User-supplied pattern
	3	Advance and backspace records
	4	Advance and backspace files
CRD† (Card Reader/Punch)	1	Punch random data
	2	AAA_{16}, $555A_{16}$, $A555_{16}$ pattern punch
	3	Punch user-supplied pattern
	4	Punch sync check
	5	Read random data
	6	Read $AAA5_{16}$, $555A_{16}$, $A555_{16}$ pattern
	7	Read user-supplied pattern
	8	Read sync check
CD1	1	Worst case pattern (9555_{16}, $6AAA_{16}$, $5A5A_{16}$, $A5A5_{16}$)
CD2	2	All ones
DK1	3	Random data
DK2	4	Random data, random block length
(Disk)	5	Zeros written over ones
	6	Random sector addresses

†Sections of this test must be requested individually.

TABLE 7-1. SCMM TEST SECTIONS (Contd)

Test	Section	Description
DRM (Drum)	1 to 6 7	Same as CD1 and DK1 Write sector number
DM1 (Drum)	1 to 6 7	Same as CD1 and DK1 Track switching test
PTR† PR1 (Paper Tape Reader)	1 2 3 4	Read random data Read $AAA5_{16}$, $55AA_{16}$, $A555_{16}$ pattern Read user-supplied pattern Read sync check
PTP† PP1 (Paper Tape Punch)	1 2 3 4	Punch random data Punch $AAA5_{16}$, $55AA_{16}$, $A555_{16}$ pattern Punch user input pattern Punch sync check
405† (Card Reader)	1 2 3 4	Read random data Read $AAA5_{16}$, $55AA_{16}$, $A555_{16}$ pattern Read user input pattern Read sync check
DVP† CVP† (Disk)	1 2	Variable positioning test Two-position seek test
AD1 AD2 (Analog-to-Digital Converter)	1	Histogram of analog input values
STU (1572-1 Timer, local mode)	1 2 3 4	Function and status of LST and SRG Transfer data to multiple register of SRG Count interrupts from LST, SRG Same as above, operator controlled
LLV	1	All bits are set to 0. FF_{16} (1555) or $FFFF_{16}$ (1553) is output.
RLY (1555/1544, 1553/1544 Digital Output/Digital Input)	2 3 4 5	All bits are set to FF_{16} or $FFFF_{16}$ and a zero is output. User pattern is output. A 1 bit is shifted across, then a 0 bit is shifted across. A 0 bit is shifted across, then a 1 bit is shifted across.
CTR (1547 Events Counter Unit)	1 2	Status, function test Counting, interrupts, and events per unit time

†Sections of this test must be requested individually.

OPERATOR ENTRY† / SCMM REPLY	COMMENTS
PRESS MANUAL INTERRUPT	
MI	
SCMM	REQUESTS SCMM EXECUTIVE
SCMM IN 12/01/75 1035 CONTROL, TEST ID	EXECUTIVE IS IN OPERATION AND REQUESTS CONTROL AND TEST PARAMETERS. DATE: DECEMBER 1, 1975 TIME: 10:35
SRT,MTT	REQUESTS MAGNETIC TAPE TEST
BEGIN MAG TAPE TEST SECTIONS,NO. OF RECDS,RUNS	TEST IS IN OPERATION AND REQUESTS OPERATIONAL PARAMETERS
2,500,8000	OPERATOR REQUESTS TEST SECTION 1, 500 RECORDS, AND INFINITE EXECUTION TIME
DLU,DENSITY	TEST REQUESTS ADDITIONAL INFORMATION
6,556 7,800 FFFF	OPERATOR ENTERS MSOS DIAGNOSTIC LOGICAL UNIT 6 AND 7, WITH THE DESIRED DENSITIES. THE LIST OF THE LOGICAL UNITS IS TERMINATED WITH THE FFFF; THE MAGNETIC TAPE TEST IS NOW IS EXECUTION.
PRESS MANUAL INTERRUPT	
MI	
SCMM	REQUESTS SCMM EXECUTIVE
CONTROL, TEST ID	EXECUTIVE REQUESTS CONTROL AND TEST PARAMETERS
SRT,PRT	REQUEST 1742x LINE PRINTER TEST.
BEGIN LINE PRINTER TEST DLU,SECTIONS,RUNS	TEST IN OPERATION AND REQUESTS OPERATIONAL PARAMETERS
14,3E,1	REQUESTS ALL TEST SECTIONS OF DIAGNOSTIC LOGICAL UNIT 14, AND 1 PASS
PRESS MANUAL INTERRUPT	
MI	
SCMM	REQUESTS SCMM EXECUTIVE
CONTROL, TEST ID	
SRT,DK1	REQUESTS 1738/1733-1 DISK TEST
BEGIN DISK 1 TEST BEWARE OF SCRATCH CONFLICT. $C1=xxxx LU,SECTIONS,BEG SEC,END SEC,RUNS	TEST IN EXECUTION AND REQUESTS OPERATIONAL PARAMETERS
26,7E,6000,7000,2	REQUESTS ALL TEST SECTIONS OF LOGICAL UNIT 26, SECTORS 6000 TO 7000 INCLUSIVE, AND 2 PASSES.
END PRINTER TEST	LINE PRINTER TEST COMPLETED

†ALL OPERATOR ENTRIES EXCEPT MANUAL INTERRUPT MUST TERMINATE WITH A CARRIAGE RETURN.

Figure 7-1. A Typical Operator/SCMM Conversation (Sheet 1 of 2)

OPERATOR ENTRY† / SCMM REPLY	COMMENTS
PRESS MANUAL INTERRUPT MI	
SCMM	REQUESTS SCMM EXECUTIVE
CONTROL, TEST ID	EXECUTIVE REQUESTS CONTROL AND TEST PARAMETERS
STP,MTT	REQUESTS TERMINATION OF MAGNETIC TAPE TEST
END MAG TAPE TEST, 0000 RUNS TAPE LU 0006, 0000 ERRORS TAPE LU 0007, 0000 ERRORS	MAGNETIC TAPE TEST TERMINATES
PRESS MANUAL INTERRUPT MI	
SCMM	REQUESTS SCMM EXECUTIVE
CONTROL, TEST ID	EXECUTIVE REQUESTS CONTROL AND TEST PARAMETERS
XIT	REQUESTS TERMINATION OF ALL TESTS AND RELEASES SCMM EXECUTIVE
END DISK 1 TEST, 0000 RUNS, 0000 ERRORS	DISK TEST TERMINATES
SCMM OUT 12/01/75 1048	SCMM EXECUTIVE RELEASES ALLOCATED CORE AND EXITS. DATE: DECEMBER 1, 1975 TIME: 10:48

†ALL OPERATOR ENTRIES EXCEPT MANUAL INTERRUPT MUST TERMINATE WITH A CARRIAGE RETURN.

Figure 7-1. A Typical Operator/SCMM Conversation (Sheet 2 of 2)

OPERATOR/SCMM CONVERSATION

Figure 7-1 is an example of a typical operator/SCMM conversation. The purpose of this sequence is to have simultaneous operation of the magnetic tape, line printer, and disk tests. The example also depicts a test which, when set into execution, terminates only upon a request by the operator. Two methods of terminating a test before normal termination are included.

ERROR MESSAGES

The following error messages may occur during operator-SCMM interface:

PROGRAM SCHEDULED†	The program requested by the operator is already in operation.
PROGRAM NOT SCHEDULED†	The operator requested a control statement (STP, PRM, NPT, or PRT) for a test that had not been set into execution.
CONTROL ERROR†	An illegal control statement was entered by the operator.
NOT IN LIBRARY†	The test required is not in the program library.
PROGRAM STACK FULL†	No new test may be requested until one or more of the current tests is completed.
DISK ERROR	A disk error occurred during the transfer of a test from mass storage to core. The test may request parameters, or SCMM may recycle. If parameters are requested, the prudent procedure would be to abort the test by typing in ? and re-requesting the test via the SCMM monitor.

†These errors cause SCMM to recycle to the query line.

The MSOS engineering file is provided to preserve driver error information for system maintenance. The engineering file is divided into three sections:

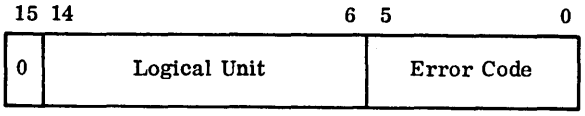
- Failure data formatting and collection
- Failure storage on mass memory
- Failure listing

DEVICE FAILURE HANDLING

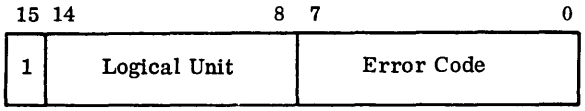
When an I/O driver determines that an error condition has occurred, it reports the error to the error logging routine EFDATA. The entry is:

```
RTJ+ LOG
```

In the Q register, two formats are used. If bit 15 is 0, Q is interpreted as:

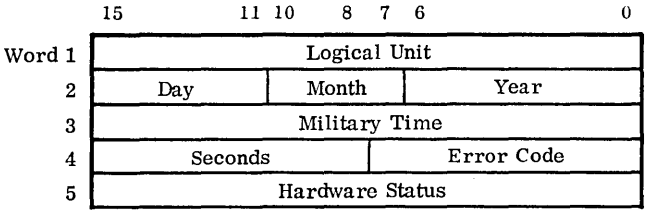


If bit 15 is 1, Q is interpreted as:



The I register contains the driver physical device table address.

The current date and time and hardware status (PDT word 12) relevant to the failure is added to this data. If the failure is on a non-mass-memory device, the data is saved in a push-down pointer table, which has a capacity of five 5-word entries. The entries are of the following form:



When the first failure is placed in the table, the mass memory failure storage module is scheduled to move the failure data to mass memory. Return is made to the driver caller. The LOG sequence is re-entrant.

If the failure is on a mass-memory device, except for the flexible disk, the failure is saved in a 10-entry push-down table. Each 5-word entry of the table is of the same form as above. This failure data is saved in core on the premise that mass memory is not reliable because of the failure.

For mass memory failures, EFDATA also logs the failure on the system comment output device with the message:

```
MM ERR xx LU=yy T=hhmm:ss S=zzzz
```

Where: xx is the error code.

yy is the logical unit.

hh is the hour.

mm is the minute.

ss is the second.

zzzz is the hardware status.

If the failure occurred during a system directory program read, the allocated space is released. The mass memory diagnostic section of EFDATA operates on the mass memory failure table at a low priority after return to the driver.

DEVICE FAILURE STORAGE

The System Initializer defines a mass-memory area of 99 sectors (preset to zero) for the engineering file. The address of this area is found in word 19 of the extended core table. One sector is allocated for each possible system logical unit. For each logical unit, 24 failures are saved as 4-word entries in a push-down/fall-off table. The first entry is the most recent failure.

The EFDATA program schedules the EFSTOR program from the system library to log failures for non-mass-memory devices. The EFSTOR program takes data from the 5-entry failure table and moves it to mass memory in the engineering file area for the failed logical unit. Each table entry is composed of words 2 through 5 of the core-resident table. The 5-entry holding table is examined and errors processed until all entries are zero.

The core-resident, mass-memory device failure table is examined for new entries, which are moved to the engineering file. Following a mass-memory error, the system is operating if further system I/O errors can be logged.

If the 5-entry holding table overflows, a diagnostic message is printed and the system must be restarted with an autoload.

DEVICE FAILURE LISTING

The device failure listing program, EFLIST, is entered via the following MIPRO mnemonic codes:

- EF – Lists all engineering file data for the system logical units. The EF entry has the format shown in figure 8-1.
- EFLU – Lists engineering file data for a specified logical unit. The EFLU entry requesting the logical unit number is:

  ENTER LOGICAL UNIT (xx)

  Where: xx is the specified logical unit (printed in the same format as EF).
- EFMM – Lists data from the core-resident, mass-memory failure table for all errors in the table. This entry produces a listing in the same format as EF, but contains only failures from the core-resident, mass-memory failure table.

LOGICAL UNIT 01 CORE ALLOCATOR

DATE	TIME	FAILURE CODE	HARDWARE STATUS

LOGICAL UNIT 02 SOFTWARE DUMMY DEVICE

DATE	TIME	FAILURE CODE	HARDWARE STATUS

LOGICAL UNIT 03 SOFTWARE DUMMY DEVICE

DATE	TIME	FAILURE CODE	HARDWARE STATUS

LOGICAL UNIT 04 713-10/711-100/713-120 CRT/SLAVE PRINTER

DATE	TIME	FAILURE CODE	HARDWARE STATUS
29 JAN 74	1903:38	00	0611
29 JAN 74	1902:15	00	0611
29 JAN 74	1901:28	00	0611

Figure 8-1. Engineering File Information Listing

The CYBER 18/1700 MSOS batch processing subsystem initiates, monitors, and terminates all jobs executed in unprotected core.

This subsystem is scheduled for execution by the operator who must manually interrupt MSOS and type:

*BATCH
or *BATCH, lu

Upon recognition of the *BATCH statement, the batch processing subsystem is scheduled to begin processing user jobs. Processing continues until an *Z control statement is encountered, at which time all job processing is terminated and the batch processing subsystem is released from core.

Jobs that the batch processing subsystem can recognize consist of all processing features executable through the use of the available job control statements. Any job to be initiated for execution must have an *JOB card as its first control statement and must be terminated by a device-detectable end-of-file.† Within a job, all legal control statements are recognized and executed by the subsystem. All legal batch processing control statements (except *V, *R, and *Z) are permissible within the boundaries defined by an *JOB and end-of-file statement.

This job structure permits a continuous flow of jobs through the subsystem without individual job initiation by operator intervention.

In the event of an abnormal job termination, all open job files are closed and the subsystem proceeds to the next job. If batch recovery is indicated by an *SR control statement, the recovery module is executed prior to job termination. This procedure is executed for all job processing errors before the next job is initiated.

If the control statement input device is the standard comment device, the character J is output, indicating that the subsystem is waiting for a new control statement.

The *BATCH,lu statement assigns the control statement input from the specified logical unit lu.

JOB CONTROL STATEMENTS

Control statements to the subsystem are format records in ASCII mode. A maximum of 72 characters is allowed for each control statement. The first character of an input statement must be an asterisk, and the last must be a blank or a carriage return if input is on the teletypewriter. Intervening characters identify the type of statement and action.

The set of legal control statements for the subsystem can be categorized as follows:

- Control statements acceptable within a job

  -Basic set

*JOB	*X	*REW	*EOF	*BSF
*V††	*LGO	*UNL	*ADR	
*U	*B	*CTO	*BSR	
*L	*SR	*PAUS	*ADF	

  -User-supplied

  *1
  *2
  *3

  -Mass storage job file handling

*DEFINE	*MODIFY
*OPEN	*FILTBL
*CLOSE	*PURGE

- Control statements acceptable to both a job and the manual interrupt routine:

  *Z, *R — May be entered at any time after manual interrupt

  *K, *CSY — May be entered only after a JOB control statement has been entered

- Control statements for loader response during a job:

  *
  *E
  *T

CONTROL STATEMENTS WITHIN A JOB

*JOB Statements

An *JOB statement instructs the subsystem to begin accepting a new sequence of control statements. It must be the first control statement in a job, and only one is allowed for each job. The date and the information on the *JOB card is printed on the list device.

†For teletypewriter input, a pseudo end-of-file (*G statement) is recognized by the job processor.

††*V,lu is also allowed outside the boundaries of a job to permit the user to initiate input from a device other than standard input.

NOTE

Whenever an *JOB statement is executed, the logical unit numbers of the standard input ($F9_{16}$), binary output (FA_{16}), and print output (FB_{16}) devices are reset to their values at autoload.

The control statement format is:

*JOB

or

*JOB,n,u,i

Where: n is the job name. The first six characters are used by the job processor. When a line printer is used, N is printed in large letters (except special characters) up to the first blank.

u is the user identification. The first six characters are saved by the job processor (required if n is used).

i is the comments (optional).

When an *JOB card is detected, the system initializes the values of standard list, standard input, and standard punch as they appear at autoload time.

*V Statement

An *V statement directs the subsystem to read all subsequent control statements from the specified logical unit.

The control statement format is:

*V,lu,m

Where: lu is the logical unit number. If not specified, the standard input is assumed.

m is the mode in which control statements are read.

A or blank	Formatted ASCII
B	Formatted binary

*U Statement

An *U statement directs the subsystem to read all subsequent control statements from the comment device. A printout at the comment device indicates that the job processor is ready to receive statements. An *U statement may occur in any order with respect to other statements within a job.

The control statement format is:

*U

*L Statement

An *L statement instructs the subsystem to call on the loader to load relocatable binary information. Once initiated, the loader continues loading from each specified logical unit until it reads an EOL block or a system control statement. The EOL block is an *T, which occupies the first two character positions. The subsystem can load from multiple logical units.

The control statement format is:

*L,lu_1,lu_2,lu_n

Where: lu_1 is the logical unit number for the loading device.

If the unit is not specified for loader input, the standard input device is used. The loader keeps track of the upper and lower limits of available core and adjusts limits according to the amount of core allocated during input.

*X Statement

An *X statement instructs the subsystem to begin program execution.

The control statement format is:

*X,N

Where: N is the memory map instruction.

Omitted	The loader is directed to produce a memory map after loading.
Specified	No memory map is produced.

When this statement is executed, the loader detects any unpatched externals and searches the program directory for a matching name. For each one found, the library routine is loaded into unprotected core as part of the job.

If an unpatched external does not match any name in the program directory, the loader comments with an E on the standard comment device. When all unpatched externals are printed on the standard print device, the loader interrogates the comment device for an *, *E, or *T statement.

*	Causes execution, regardless of unpatched externals
*E	Directory of part 0 core-resident entry points is searched for missing names. If externals are still undefined, the loader interrogates the comment device for an * or *T statement.
*T	Causes job termination

After patching all externals, the loader returns to the subsystem that tests the breakpoint switch. If this switch is off, the subsystem immediately executes the user's program. If it is set, the breakpoint routine is loaded and the subsystem enters the breakpoint routine.

*LGO Statement

*LGO is the load-and-go command; the subsystem calls the loader to load relocatable binary programs. The loader loads from each specified logical unit until it reads an EOL block or a system control statement. Loading may occur from multiple logical units.

The control statement format is:

*LGO,N,lu_1,lu_2, . . . ,lu_{10}

Where: N is specified. No memory map is produced.

lu is the logical unit number for the loading device. If no parameters are specified (*LGO,N), the standard scratch device is used.

or

*LGO,lu_1,lu_2, . . . ,lu_{10}

Where: lu_1,lu_2,...,lu_{10} is the logical unit number for the loading device; a memory map is produced. If no parameters are specified (LGO), the standard scratch device is used.

The loader keeps track of the upper and lower limits of available core and adjusts the limits according to the amount of core allocated during input.

When the go portion of this statement is executed, the loader detects any unpatched externals and searches the program directory for a matching name. For each one found, the library routine is loaded into unprotected core as part of the job.

If an unpatched external does not match any name in the directory, the loader comments with an E on the standard comment device. When all unpatched externals are printed on the standard print device, the loader interrogates the comment device for an *, *E, or *T statement.

After patching all externals, the loader returns to the subsystem that tests the breakpoint switch. If this switch is off, the user's program is executed. If it is set, the breakpoint routine is entered.

The load-and-go option (using *L, *X, or *LGO statements) provides for execution immediately following compilation or assembly.

When load-and-go output is specified to the compiler or assembler, binary output is placed on the scratch unit, starting at sector 1 of the scratch area. The compiler or assembler produces an EOL statement for the end of binary output and stores the end of the load-and-go block count in $E4_{16}$. The binary output of the next compilation or assembly begins at the sector containing the EOL and continues until the assembly or compilation is completed. Enter an *LGO statement to load binary information from the scratch unit.

For example, if the library is on logical unit 8 and programs are to be compiled, assembled, loaded, and executed, the following statements are required:

*FTN	Load and execute FORTRAN; one of the parameters to FORTRAN requests load-and-go
*ASSEM	Load and execute the assembler; one of the parameters to the assembler requests load-and-go
*LGO	Load load-and-go information and execute the program just loaded.

*B Statement

An *B statement instructs the subsystem to turn on the breakpoint load switch. It may occur in any order with respect to other statements wihtin a job.

The control statement format is:

*B

The breakpoint program is stored in the system library and must be loaded before execution of the job. If the breakpoint load switch is on when an *X or *LGO statement is processed, the subsystem loads the breakpoint routine into unprotected core with the job to be executed. If there is insufficient core, the error message

E5
E10 BRKPT

is printed on the standard list device and the job is executed without the requested breakpoints.

The operator can then use the breakpint program for the following functions:

- Set breakpoint addresses in the job
- Make entries in the A, Q, and I registers
- Make entries into unprotected core
- Execute a transfer or a return transfer to unprotected core
- Take core dumps
- Take mass storage dumps
- Print the register contents

Breakpoint control statements are entered on the comment device. When the operator types *C in response to a breakpoint program query, the job is re-entered from the breakpoint program. Additional information on breakpoint programs is given in Breakpoint Program, section 10.

For example, if a job is to be loaded from logical unit 7, and the breakpoint program is to be loaded with the job, the following statements are required:

*JOB	
*B	Set the breakpoint load switch.
*L,7	Load the job from logical unit 7.
*X	Complete the load by patching the remaining externals from the program library, load the breakpoint program, and execute. At this time, BP on the comment device indicates the breakpoint program is waiting for a control statement.

*SR Statement

An *SR statement instructs the job subsystem to set the recovery indicator. Upon termination of job execution, the recovery program is entered. A message, RE, on the comment device indicates that the recovery program has been entered.

The control statement format is:

*SR

When the recovery indicator is set, the recovery program is entered when the end-of-file for the job is encountered (*G on the teletypewriter). This occurs if the recovery indicator is set when the job terminates normally or abnormally.

When the recovery program is terminated by the operator typing an *T recovery program control statement, the subsystem continues processing jobs.

When the recovery program is in core, the operator may dump core and mass storage. All input to the recovery program is from the comment device (refer to Recovery Program, section 10).

*REW Statement

The *REW statement instructs the subsystem to rewind the specified logical units to their loadpoint. Several logical units can be specified in one request. Up to five logical units may be specified in an *REW statement. An error message is used if the number exceeds five.

The format is:

*REW,lu_1,lu_2,lu_3,lu_4,lu_5

*UNL Statement

An *UNL statement instructs the subsystem to rewind and unload the specified logical unit. Up to five logical units may be specified in an *UNL statement.

The format is:

*UNL,lu_1,lu_2,lu_3,lu_4,lu_5

*ADR Statement

The *ADR statement instructs the subsystem to advance a specified number of records on a specified logical unit.

The format is:

*ADR,lu,nn

Where: lu is the logical unit number of the device.

nn is the decimal number of records to advance in the range 1 through 32,767.

*BSR Statement

The *BSR statement instructs the subsystem to backspace a specified number of records on a specified logical unit.

The format is:

*BSR,lu,nn

Where: lu is the logical unit number of the device.

nn is the decimal number of records to backspace in the range 1 through 32,767.

*ADF Statement

The *ADF statement instructs the subsystem to advance a specified number of files on a specified logical unit.

The format is:

*ADF,lu,nn

Where: lu is the logical unit number of the device.

nn is the decimal number of files to advance in the range 1 through 32,767.

*BSF Statement

The *BSF statement instructs the subsystem to backspace a specified number of files on a specified logical unit.

The format is:

*BSF,lu,nn

Where: lu is the logical unit number of the device.

nn is the decimal number of files to backspace in the range 1 through 32,767.

*CTO Statement

An *CTO statement causes the comments appearing on the card to be printed on the standard comment device for operator information. Continuation cards are not allowed. The format is:

*CTO,comments

*PAUS Statement

This is the pause indicator; the format is:

*PAUS

On encountering an *PAUS, READY? is typed on the standard comment device. The next control statement is read only after a carriage return reply to this message. This statement can be used in conjunction with the CTO statement to control operations. A time-out causes the message to be typed and a new request for input to be made.

*EOF Statement

An *EOF statement instructs the subsystem to write one end-of-file to the current standard binary output device.

*Control Statement

An * control statement resets the load-and-go pointer to one and clears the loader-in-core flag. When load-and-go mass storage area is to be used by more than one program in a job, this statement should be used to ensure proper execution. For example, when used after a call to the assembler or compiler, the next assembled or compiled program replaces (rather than adding to) the last program (when x option is used).

An * statement should be entered to clear the loader-in-core flag whenever load operations have been performed, but a program has not executed, and the user wishes to initiate a new load sequence from the start of unprotected core.

*Entry Point Name Statement

An *entry point name statement instructs the subsystem to call on the loader to load a program from the program library. The entry point name must appear in the program library directory.

The format is:

*entry point name

The program, which is stored in relocatable binary format in the program library, is loaded into available core. The loader records the limits of available core before and after the load.

This operation is a program load. A program loaded into core by a program load is entered immediately for execution. An *X statement is not needed.

Example:

*FTN Load a program from the program library with the entry point FTN.

End-Of-File Card

This is either a user-supplied or a device-detectable code that is used to terminate a job. It must be the last control statement in a job. For card readers, a 6/7/8/9 sequence in column 1 is used. For the teletypewriter input, an *G control statement serves the function of an end-of-file. Refer to the appropriate CYBER 18/1700 peripheral equipment reference manual for end-of-file capabilities for specific devices.

USER-SUPPLIED STATEMENTS

User-supplied statements are:

*1, *2, or *3

These statements are subroutines with entry points ONE, TWO, and THREE and are declared as relative externals in the statement analyzer module.

Since these statements are supplied by the user, no attempt is made to define their formats, uses, or functions.

MASS STORAGE JOB FILE HANDLING STATEMENTS

The installation of the file manager package and the pseudo tape driver in the operating system provide the capability to create, use, and delete special job files. The following statements are used to define files and connect these files to an appropriate pseudo tape device. Having established the linkage of a file to a pseudo tape, I/O may be performed on the file as though it were a magnetic tape by using the appropriate monitor request (READ, WRITE, FREAD, FWRITE, or MOTION). These files provide capabilities such as the establishment of dedicated files for binary and list output. ASCII or binary information is possible, such as that now reserved on magnetic tape for frequently executed system programs (COSY, LIBEDT, ASSEM, FTN, etc.).

The number of job files available is an option selected at installation.

The number of job processing operations that may be performed concurrently during a job is a function of the number of pseudo tape logical units in the system. For example, to input, punch, and list three files during the same job, three pseudo units are required. The number of pseudo units required is equal to the number of operations assigned to files for the same job by an *K statement.

The following control statements provide the information required by the pseudo driver for interface with the file manager and provide maximum security and use of these files.

If any of these file handling requests is rejected for any reason, a diagnostic is typed at the comment device and the subsystem terminates the job (refer to the MSOS Diagnostic Handbook).

*DEFINE Statement

An *DEFINE statement causes the job file handler to create a mass storage file with the parameters contained in the request.

The control statement format is:

*DEFINE,fileid,sc,date

Where: fileid is any one to six alphanumeric characters, of which the first character must be alphabetic.

sc is the security code of one to six alphanumeric characters.

date is the expiration date after which the file is released with an *PURGE request. If no date is supplied, the current system date is used. Calendar format (mmddyy) is used:

mm	month	(01 through 12)
dd	day	(01 through 31)
yy	year	(00 through 99)

*RELEAS Statement

An *RELEAS statement causes the job file handler to close and release a mass storage file with a matching file identification (fileid) and security code (sc).

The control statement format is:

*RELEAS,fileid,sc

Where: fileid is any one to six alphanumeric characters, of which the first character must be alphabetic.

sc is the security code of one to six alphanumeric characters.

*OPEN Statement

An *OPEN statement enables a user to reference a predefined job file for read and/or write requests.

This statement tells the job file handler to permit a user to reference a job file with the matching fileid and sc. If the file is opened with an R (read) option, the file can be read only; with the W (write) option, the file can be read and/or written by the user. When opened, the file is at its load point. Requests to open a previously opened file are rejected. The number of job files open at any one time may not exceed the number of pseudo tape units available. Only one job file may be opened on a pseudo tape logical unit at any given time.

The control statement format is:

*OPEN,fileid,sc,R/W,lu

Where: fileid is any one to six alphanumeric characters, of which the first character must be alphabetic.

sc is the security code in one to six alphanumeric characters.

R is read only.

W is read or write.

lu is the logical pseudo tape unit to which the file is assigned.

*CLOSE Statement

An *CLOSE statement causes the specified job file to be made unobtainable to a user.

This statement tells the job file handler to reject all subsequent references for read/write on a file with the specified file name (fileid) and security code (sc). After the execution of an *CLOSE statement, the specified file is rewound to its load point and any writing capability is removed. In the event of abnormal job termination, all opened files are closed for user protection.

The control statement format is:

*CLOSE,fileid,sc

Where: fileid is any one to six alphanumeric characters, of which the first character must be alphabetic.

sc is the security code in one to six alphanumeric characters.

*MODIFY Statement

An *MODIFY statement modifies the definition parameters of a defined file.

An *MODIFY enables a user to modify the parameters that define a current file with the specified old file name (fileid) and security code (sc).

The control statement format is:

*MODIFY,fileid,sc,nfileid,nsc,date

Where: fileid is the old file name in one to six alphanumeric characters, of which the first character must be alphabetic.

sc is the old security code in one to six alphanumeric characters.

nfileid is the new file name in one to six alphanumeric characters, of which the first character must be alphabetic.

nsc is the new security code in one to six alphanumeric characters.

date is the new expiration date after which the file can be released with an *PURGE request. If no data is supplied, the current system date is used. Calendar format (mmddyy) is used:

mm	month	(01 through 12)
dd	day	(01 through 31)
yy	year	(00 through 99)

*FILTBL Statement

An *FILTBL statement prints the job files currently defined. The following printout is the format for the job processor files.

name date OP/CL R/W

Where: name is the name of the file in one to six alphanumeric characters.

date is the expiration date in calendar format (mmddyy):

mm	month	(01 through 12)
dd	day	(01 through 31)
yy	year	(00 through 99)

OP/CL is the current state of the file.

OP	open
CL	closed

R/W is the current mode of the file.

R	read
W	write

*PURGE Statement

An *PURGE statement deletes a file whose expiration date is the same or less than the date specified in an *PURGE request when the proper purge key is present.

The control statement format is:

*PURGE,date,purge key

Where: date is the target expiration date in calendar format (mmddyy):

mm	month	(01 through 12)
dd	day	(01 through 31)
yy	year	(00 through 99)

purge key is an installation parameter, which the standard release system specifies as ASCII 00.

STATEMENTS ACCEPTABLE TO JOB AND MANUAL INTERRUPT ROUTINES

*K Statement

An *K statement is used to reassign standard system logical unit numbers.

With an *K statement, the operator can select devices for system units other than those currently used.

The parameters in an *K statement are not ordered, but must be separated by a comma and followed by a carriage return or a space.

The control statement format is:

*K,Ilu,Llu,Plu

Where: lu is the logical unit number (in all cases).

I is the system input unit.

L is the system print unit.

P is the system binary output unit.

One location in the communications region contains a physical device table or ordinal for each of the standard system devices. The logical unit number in an *K statement replaces the number in the communications region.

An error exit is taken if a unit number designated a protected device, or if the print unit specifies a mass storage device.

*CSY Statement

An *CSY statement reassigns standard COSY logical unit numbers. This control statement to the subsystem is for use with the COSY driver. The I, P, and L parameters may also be used to assign logical units for the COSY program control statements. If no logical units are specified on the COSY control cards, the assigned units are used.

The parameters in an *CSY statement are not ordered, but must be separated by commas and the last parameter must be followed by a carriage return or a space.

The control statement format is:

*CSY,Lxx,Iyy,Pzz

Where: xx is the logical unit of the COSY list output.

yy is the logical unit of the COSY input library.

zz is the logical unit of the Hollerith or COSY output.

Use of the COSY driver requires the following sequence of commads:

*CSY,Iaa
*K,Izz,Pcc,Lbb

Where: aa is the logical unit of the COSY input.

zz is the logical unit of the COSY driver.

cc is the logical unit of the standard punch device.

bb is the logical unit of the standard list device.

The next input statement could then be an *ASSEM or *FTN, which allows direct assembly or compilation from a COSY source.

NOTE

COSY revisions are not permitted when using the COSY driver.

*Statement

When the execution of a program in unprotected core is interrupted by a manual interrupt, typing an * causes job execution to continue. When encountered in a job stream, the * statement resets the load-and-go pointer and clears the loader-in-core flag.

*Z Statement

An *Z statement, which marks the end of batch processing, is accepted by the subsystem, regardless of the order of its appearance.

The format is:

*Z

After reading an *Z statement, the subsystem performs the following functions:

- Releases core space occupied by the subsystem
- Sets protect bits for all locations previously in unprotected core
- Releases this core area to the core allocator, which makes it available to protected system programs
- Resets the load-and-go pointer in location $E4_{16}$ to one

During the execution of a program in unprotected core, the operator may terminate a job with a manual interrupt, followed by typing an *Z. When this is done, the subsystem performs the following functions:

- Deletes interrupt stack entries referring to unprotected core
- Sets the completion of all I/O into unprotected core to the address of the dispatcher
- Waits for the completion of all I/O from unprotected core if unbuffered protect processor operations are in execution
- Waits for the completion of all timer requests from unprotected core
- The job terminates.
- Initiates a new job

*R Statement

An *R statement informs the operating system that a device that previously failed is operable and ready for input/output.

The control statement format is:

*R,lu

Where: lu is the logical unit number of the failed device.

If a device fails and an alternate device has been assigned, the alternate device is used to process requests (refer to Alternate Devices, section 2).

Example:

When a device associated with logical unit number 6 fails (and logical unit 6 has an alternate), the operator is notified by a comment and input/output processing continues on the alternate device.

When the operator has taken corrective action regarding the device that failed, he notifies the operating system:

Press MANUAL INTERRUPT

Type *R,6

This procedure restores logical unit 6 to the primary device.

LOADER RESPONSE DURING JOB EXECUTION

The following control statements are used for loader response during job execution:

*
*E
*T

These statements have been discussed briefly in this section under the *X and *LGO statements. Refer to Unprotected Program Requests, section 3, for a complete discussion of these statements.

CYBER 18/1700 MSOS has five routines to aid in debugging programs:

- On-line debug package — Protected and unprotected core
- Breakpoint program, Recovery program — Unprotected core
- Abort dump
- On-line snap dump

ON-LINE DEBUG PACKAGE

The on-line debug package (ODEBUG) allows the programmer to access both protected and unprotected core in order to change core and mass storage locations and to execute debugging functions while the system is running in an on-line state.

The program is resident on mass storage, but is executed in allocatable core. When ODEBUG is initiated, allocatable core is divided into three parts:

- Area permanently assigned to the executive program
- Area containing function processors; this may extend into the third area when necessary
- Area to which subroutines are transferred as needed

These areas are released when ODEBUG terminates.

When activated, the memory required by this package is 370_{16}. However, care must be exercised when single/double precision operations are involved. The package uses its own internal floating point arithmetic packages (single precision as well as double precision). Therefore, it makes a monitor request to allocate about 1800 words from allocatable core to bring in the proper floating point routines for value conversion. This allocatable core is released as soon as the value conversion is finished. These routines need 1800 words of allocatable core; system allocatable core can be temporarily unavailable to other requests.

OPERATOR PROCEDURES

ODEBUG is initiated by pressing MANUAL INTERRUPT on the teletypewriter and typing in the characters DB. ODEBUG alerts the operator that it is in core and ready for use:

```
DEBUG IN
```

The operator may then type in a request. Each request must be terminated by a carriage return. All requests are limited to one line of up to 80 characters on the comment device. After the request is completed and all associated messages have been typed, ODEBUG types:

```
NEXT
```

ODEBUG is then ready for the next request from the operator.

To terminate ODEBUG, the operator types:

```
OFF
```

ODEBUG types:

```
DEBUG OUT
```

Many requests that could have lengthy I/O operations can be terminated abnormally.

To terminate I/O, the operator must

```
Press MANUAL INTERRUPT
Type DX
```

The debug messages are:

```
DEBUG IN
DEBUG OUT
DB I/O ERROR
NEXT
DB FORMAT INCORRECT
DB INVALID REQUEST
CELL CONTENT
DB SEARCH FINISHED
DB NO CORE AVAILABLE
DB ILLEGAL LU
DB ILLEGAL MM ADD.
DB ORDINAL OVER MAX.
DB ORDINAL LENGTH ZERO
```

A command that causes core or mass memory data to be altered requires confirmation by the operator. The new data to be entered and the old data currently in the system are printed for referencing purposes. The operator must type OK if the change is approved. A carriage return or any characters other than OK nullifies the change.

Confirmation is also required by the move data commands, such as MBC, MMM. The first word of both new and old data is printed for reference. OK is entered by the operator for confirmation. Any other entry causes the request to be aborted.

DATA INPUT REPRESENTATION

The mnemonic symbols used throughout this section are summarized as follows:

- All data values are subscripted to designate the input values:

  h_h Hexadecimal value entry; a maximum of four hexadecimal characters

  i_n Decimal integer value entry; a signed 5-digit value

  a_n ASCII character entry; a string of ASCII characters

  s_n Single-precision value entry; both Fw.d and Ew.d formats are allowed. The Fw.d format has a maximum of seven digits for the numerical part and is signed if needed. The Ew.d format has a maximum of seven digits for the numerical part and two digits for the exponent part. Both numerical and exponents parts can be signed.

  d_n Double-precision value entry; both Fw.d and Dw.d formats are allowed. A signed 12-digit number is the maximum value for the Fw.d format, while the maximum digits for the numerical and exponent parts are 12 and 2, respectively, for the Dw.d format.

  Examples for s_n and d_n are shown below:

	Value	Equivalent Representation
s_n	+12.34	12.34000 = .1234E2 = 1.234E + 01 = 0.123400E2 = 1.23400E + 1 = 1234E-02
d_n	+12.34	12.3400000000 = 0000000012.34 = .1234E2 = +.1234000000000E + 02 = 1234E-02 = 1234E-2
s_n	-0.00056	-0.000560 = -.0056E -01 = -.000056E + 1 = -.56E-3
d_n	-0.00056	-0.00056000000 = -56E-05 = -5.6000000000E-04 = -56E-5

  The above data value entries are separated by commas and terminated by a carriage return. However, no delimiter (comma) is required for the ASCII data entry. The ASCII entry is a string of ASCII characters terminated by a carriage return.

- The parameters for core data representation are:

  sc The beginning core location (hexadecimal value)

  ec The end core location (hexadecimal value)

  b The base (hexadecimal value); 0 is assumed when unspecified

  nsc The beginning core location of block 2 (hexadecimal value)

  scr The start of core location to be released (hexadecimal value)

  sa The scheduled address (hexadecimal value)

- Mass Memory Representative

  Three parameters are used for mass memory sector addressing entries. The first two parameters are for the most significant bits and least significant bits sector representation, and the third parameter is for a word within a sector. Note that the third entry value is not limited to the range of 0 through 95 (one sector). All three entries must be entered and are in hexadecimal mode.

  ssmsb The most significant bits of the starting sector (0 to 1)

  sslsb The least significant bits of the starting sector (0 to 7FFFF)

  sw The starting word

  nsmsb The most significant bits of the new starting sector

  nslsb The least significant bits of the new starting sector

  nsw The new starting word

  Sector value is expressed in most significant bits and least significant bits to account for mass storage sizes of 4.5 million words or more.

  lu The logical unit (decimal value)

  nlu The new logical unit (decimal value)

  Two logical units are available for system with multiple mass storage units:

  wa The word addressing mass memory address (hexadecimal value)

  All mass memory addresses are checked for maximum storage capacity within this package by means of their equipment type codes.

- The parameter for magnetic tape is:

  de The density (decimal value)

- The parameters for miscellaneous representations are:

  ord The ordinal number (decimal value)

  nw The number of words (hexadecimal value)

  p The bit pattern (hexadedimal value)

x	The logical unit (decimal value)
m	The bit mask for search matching (hexadecimal value)
i	The increment (hexadecimal value)
q	The passed Q-register data (hexadecimal value)
pl	The scheduled priority level (hexadecimal value)
length	The request size (hexadecimal value)
rp	The request priority (hexadecimal value)
ltt	The location of the top of thread (hexadecimal value)
mod	The mode for output format (ASCII character)
prt	The part 1 request indicator (0 or 1)

DEBUG MAINFRAME REQUESTS

The format to store data in core is:

LHX,SC,b/h_1,h_2,h_3 . . . h_n

Examples:

Store data into core location + base:

LHX,1601,0/C8*1606 (Stores 14EA into location 1600)

One-word relative instructions:

LHX,1601,0/C8*1606 (Stores C805 into location 1601)

Core is loaded with the 2-digit OP code preceding the asterisk and the 8-bit relative increment that is obtained by subtracting the address of the core location in which the data is to be stored from the address that follows this asterisk. The relative increment must be less than ±127; otherwise, an incorrect relative increment is stored.

16-Bit relative address:

LHX,1602,0/*1703 (Stores 0101 into location 1602)

Core is loaded with the 16-bit relative increment obtained by subtracting the address of the core location in which the data is to be stored from the address which follows this asterisk.

NOTE

LHX,1600,0/14EA,C8*1606,*1703 has the same result as the three preceding examples.

Four types of input data can be entered:

- Decimal – LIT,SC,b/i_1,i_2,i_3, . . .i_n
- ASCII character – LAS,SC,b/a_1,a_2,a_3,a_4, . . .a_n
- Single precision – LSP,SC,b/s_1,s_2, . . . s_n
- Double precision – LDP,SC,b/d_1,d_2,d_3, . . . d_n

NOTE

The single/double precision data is converted to the two-word/three-word floating value format before being stored in core. For a detailed description of the single/double precision format, refer to the MSOS FORTRAN Reference Manual. Only the last four digits of hexadecimal data entries and the last five digits of decimal data entries are accepted when more than the above numbers are entered.

DUMP DATA FROM CORE

Data output can be requested in one of five formats: hexadecimal, decimal, ASCII characters, and single- or double-precision floating point. The current core location is printed on the left side of each line of the output data. The listing unit can be specified by the user via the change logical unit command. The standard comment unit is normally used unless the user redefines the list unit. The following are the command formats.

Data Form	Format	Description
Dump hexadecimal	DPC,sc,ec,b	Print the 4-digit hexadecimal values for the request core locations.
Dump decimal (integer)	DIC,sc,ec,b	Print the signed 5-digit decimal values for the requested locations.
Dump ASCII	DAS,sc,ec,b	Print sets of two ASCII characters with a space separating each set.
Dump single precision	DSP,sc,ec,b	Print single-precision value(s) of seven significant digits in an Ew.d format as shown: (sign)0.xxxxxxxxxxxxE(Sign)xx Where x represents the integer value digit.

Data Form	Format	Description
Dump single precision (contd)	DSP,sc,ec,b	The number of digits representing the mantissa and exponent parts are seven and two, respectively. For overflow or underflow conditions, the number is represented by a string of 14 asterisk characters.
Dump double precision	DDP,sc,ec,b	Print the double-precision value of 12 significant digits in Dw.d format as shown: (sign)0.xxxxxxxxxxxxd(sign)xx The number of digits representing the mantissa and exponent is 12 and two, respectively. A string of 17 asterisk (*) characters is used to represent either underflow or overflow value.

TRANSFER DATA CORE TO MASS MEMORY

The following are the formats used to transfer data core to mass memory:

Data Form	Format	Description
Write operation	WCD,ssmsb,sslsb, sw,sc,nw WDK,sc,ec,ssmsb, sslsb,sw	Transfer data from core to mass memory.
Read operation	RDC,ssmsb,sslsb, sw,sc,nw RDK,sc,ec,ssmsb, sslsb,sw	Transfer data from mass memory to core.
Set mass memory to pattern	SMP,ssmsb,sslsb, sw,nw,p	Set mass memory to a specified bit pattern.

LOGICAL UNIT ALTERATION

The list unit can be respecified by the operator with the following command:

CLU,x

It should be noted that the output of certain commands (table 10-1) is not affected by this command.

If more than one mass memory unit exits, they can be redefined by using the following command:

MLU,x

Note that these new logical unit definitions remain in effect for debug usage until it is either respecified or the ODEBUG function is terminated.

TABLE 10-1. ODEBUG COMMANDS

Command	Confirmation	List Logical Unit	Mass Memory Logical Unit	Input Logical Unit	DX	Comments
LHX	Yes	Comment device	†	Comment device	No	
LIT	Yes	Comment device	†	Comment device	No	
LAS	Yes	Comment device	†	Comment device	No	
LSP	Yes	Comment device	†	Comment device	No	
LDP	Yes	Comment device	†	Comment device	No	
DPC	No	Comment device/ user option	†	Comment device	Yes	
DIC	No	Comment device/ user option	†	Comment device	Yes	
DAS	No	Comment device/ user option	†	Comment device	Yes	

†Not applicable

TABLE 10-1. ODEBUG COMMANDS (Contd)

Command	Confirmation	List Logical Unit	Mass Memory Logical Unit	Input Logical Unit	DX	Comments
DSP	No	Comment device/ user option	†	Comment device	Yes	
DDP	No	Comment device/ user option	†	Comment device	Yes	
WCD	No	†	Library unit/ user option	Comment device	No	
WDK	No	†	Library unit/ user option	Comment device	No	
RDC	No	†	Library unit/ user option	Comment device	No	
RDK	No	†	Library unit/ user option	Comment device	No	
LST	No	Comment device	†	Comment device	No	
SMP	No	†	Library unit/ user option	Comment device	No	
CLU	No	†		Comment device	No	
MLU	No	†		Comment device	No	
SCN	No	Comment device	†	Comment device	Yes	
SPE	No	Comment device	†	Comment device	No	
ADH	No	Comment device	†	Comment device	No	
SBH	No	Comment device	†	Comment device	No	
SET	No	Comment device	†	Comment device	No	
SPP	No	Comment device	†	Comment device	No	
CPP	No	Comment device	†	Comment device	No	
MBC	Yes	Comment	†	Comment	No	Print first word of both new and old address.
CCC	No	Comment device/ user option	†	Comment device	Yes	
SCH	Yes	Comment device	†	Comment device	No	Print the contents of the first word of the scheduled program.
ALC	No	Comment device	†	Comment device	No	
REL	No	Comment device	†	Comment device	No	
DAC	No	Comment device/ user option	†	Comment device	No	
DPT	No	Comment device/ user option	†	Comment device	No	

†Not applicable

TABLE 10-1. ODEBUG COMMANDS (Contd)

Command	Confirmation	List Logical Unit	Mass Memory Logical Unit	Input Logical Unit	DX	Comments
PTH	No	Comment device/ user option	†	Comment device	No	
MSD	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
MMM	Yes	Comment device	User option	Comment device	Yes	Print the first word of both the new and old address.
DMH	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
DMI	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
DMA	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
DMS	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
DMD	No	Comment device user option	Library unit/ user option	Comment device	Yes	
CWA	No	Comment device	†	Comment device	No	
CCM	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
CMM	No	Comment device/ user option	User option	Comment device	Yes	
SMN	No	Comment device/ user option	Library unit/ user option	Comment device	Yes	
ADF	No	Comment device	†	Comment device	Yes	
BSF	No	Comment device	†	Comment device	Yes	
ADR	No	Comment device	†	Comment device	Yes	
BSR	No	Comment device	†	Comment device	Yes	
WEF	No	Comment device	†	Comment device	Yes	
REW	No	Comment device	†	Comment device	No	
UNL	No	Comment device	†	Comment device	No	
SLD	No	Comment device	†	Comment device	No	
LHC	Yes	Comment device	Library unit	Comment device	No	
LIC	Yes	Comment device	Library unit	Comment device	No	
LAC	Yes	Comment device	Library unit	Comment device	No	
LHO	Yes	Comment device	Library unit	Comment device	No	

† Not applicable

TABLE 10-1. ODEBUG COMMANDS (Contd)

Command	Confirmation	List Logical Unit	Mass Memory Logical Unit	Input Logical Unit	DX	Comments
LIO	Yes	Comment device	Library unit	Comment device	No	
LAO	Yes	Comment device	Library unit	Comment device	No	
LSO	Yes	Comment device	Library unit	Comment device	No	
LDO	Yes	Comment device	Library unit	Comment device	No	
LHM	Yes	Comment device	Library unit/ user option	Comment device	No	
LIM	Yes	Comment device	Library unit/ user option	Comment device	No	
LAM	Yes	Comment device	Library unit/ user option	Comment device	No	
LSM	Yes	Comment device	Library unit/ user option	Comment device	No	
LDM	Yes	Comment device	Library unit/ user option	Comment device	No	

†Not applicable

GENERAL CPU OPERATIONS

Core locations are searched from start core to end core by the increment for a match between AND (mask, number) and AND (mask, core). The format is:

SCN,sc,ec,n,m,i

An example of this operation is:

SCN,0,7FFF,A1F7,FF00,2

Where 0,2,4,6,8, . . . are searched for AND resulting in A1xx, where xx may have any value. The locations containing the searched configuration and their contents are printed after the following heading:

CELL CONTENTS

The following are formats for CPU operations:

Data Form	Format	Description
Search core for parity error	SPE,ec	Print the location of the parity error. The following comment terminates the request: SEARCH FINISHED
Add hexadecimal numbers	ADH,number1, number2, . . . number8	Add up to eight hexadecimal numbers.
Subtract hexadecimal numbers	SBH,number1, number2	Subtract number 2 from number 1, in hexadecimal.
Set core	SET,sc,ec,p	Set core with a pattern from start to end.
Set program protect bit	SPP,sc,ec	Set the program protect bits.
Clear program protect bit	CPP,sc,ec	Clear the program protect bits.
Move blocks of core	MBC,sc,ec,nsc	Move a block of core from one location to another. Before moving the block, the first words of the new and the old data are printed for operator verification. This allows the operator to abort the current selection if any errors in defining the parameters are discovered.
Compare core to core	CCC,sc,ec,nsc	Compare the core contents of data in block 1 (sc) and block 2 (nsc). The differences are printed under the heading CELL CONTENT. Both the locations and contents of blocks 1 and 2 are

Data Form	Format	Description
Compare core to core (contd)	CCC,sc,ec,nsc	printed. The following is an example: CELL CONTENT 05F6 (1111) 58AC (0111) Where: 05F6 is the current sc location. (1111) is the content. 58AC is the current nsc location. (0111) is the content.

MONITOR OPERATIONS

Schedule Program

A specified core location can be scheduled for execution at the specified priority level, with the user passing a parameter contained in the Q register. The contents of the scheduled location are printed. If it contains the proper data, the operator can confirm (or approve) the schedule request by typing in OK. Conversely, this request can be aborted by entering any character(s) other than OK. The format is:

SCH,sa,q,pl,prt

Allocatable Core

The format of the command to generate an area within allocatable core is:

ALC,length,rp

Where: length is hexadecimal.

rp is the request priority (the minimum priority is three) of the location of allocated core that is printed out as follows:

CORE ALLOCATED FROM $h_1h_1h_1h_1h_1$ to $h_2h_2h_2h_2h_2$

$h_1h_1h_1h_1h_1$ is the start of the allocated core area.

$h_2h_2h_2h_2h_2$ is the end of allocated core.

Release Core

The format is release allocated core is:

REL,SCR

List Allocatable Core Map

The dump allocatable core command is:

DAC

The following printout represents the core configuration.

LLLL and MMMM are lengths of two programs in core starting at xxxx and zzzz, respectively.

Printout

ALLOCATABLE CORE MAP

xxxx LLLL

yyyy

(indicates empty core)

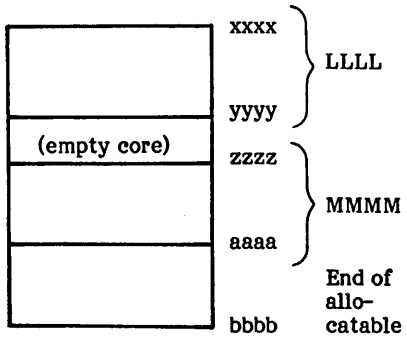
zzzz MMMM

aaaa

* (indicates empty core)

bbbb

Core Configuration



ListPartition Core Map

The format for this request is:

DPT

If partition core is being used, the printout for this command is as follows:

PARTITION CORE MAP

PARTITION NO. nn llll mmmm

Where: nn is the partition number.

llll is the core location.

mmmm is the size.

If partition core is unused, the printout for this command is as follows:

PARTITION CORE MAP

UNUSED

Print Thread

The print thread request format is:

PTH,ltt,b

Where: ltt is the location of top of thread.

b is the base.

The format for each entry printout is as follows:

loc,w_1,w_2

Where: loc is the location.

w_1 is word 1 of the request.

w_2 is word 2 of the request.

The end-of-thread indicator (FFFF) terminates the printout, or if there are more than ten entries on the thread, only the first ten are printed. If the thread is empty, the printout consists of the end-of-thread indicator only. A sample printout is:

```
153A,   120A,   2357
087B,   1277,   4324
44AC,   1204,   803A,
FFFF
```

Interrupts are inhibited while the thread is being searched. The priority level of the dump is the same as ODEBUG.

List All Debug Commands

All command mnemonics are printed on the standard list device. LST is the debug command.

MAGNETIC TAPE OPERATIONS

The following requests perform tape motion commands. The logical unit number is specified by lu. The parameters of these requests are decimal. If the number of iterations is blank, one iteration is assumed.

NOTE

The number of records/files is limited to 4095 for all magnetic tape requests. Other standard device drivers accept the ODEBUG magnetic tape requests for motion, but only one motion function is performed (that is, single file record skips, etc.).

The following are the tape motion command formats:

Data Form	Format
Advance files	ADF,lu,number of files
Backspace files	BSF,lu,number of files
Advance records	ADR,lu,number of records
Backspace records	BSR,lu,number of records
Write end-of-file	WEF,lu,number of files
Rewind tape	REW,lu
Unload tape	UNL,lu
Select density	SLD,lu,de

Where: de is a designated density:

- 0 200 bits per inch
- 1 556 bits per inch
- 2 800 bits per inch
- 3 1600 bits per inch

Omitted 1 or 556 bits per inch

MASS STORAGE

List Mass Memory

This statement performs a dump off mass memory to list unit from the starting sector to the ending sector. The format is:

MSD,ssmsb,sslsb,esmsb,eslsb,mod

Where: mod is the mode. If not specified, data is dumped in hexadecimal mode.

- A ASCII; data is dumped directly to the liit output device.
- H Hexadecimal; data is converted to ASCII before being dumped.
- I Decimal; data is converted to ASCII before being dumped.

Dump Mass Memory

Mass memory data can be dumped in one of the following five formats:

- Hexadecimal – DMH,ssmsb,sslsb,sw,nw
- Decimal – DMI,ssmsb,sslsb,sw,nw
- ASCII – DMA,ssmsb,sslsb,sw,nw
- Single Precision – DMS,ssmsb,sslsb,sw,nw
- Double Precision – DMD,ssmsb,sslsb,sw,nw

The data formats are the same as those described in the section entitled Dump Data From Core. The output format contains the mass memory address and data. The hexadecimal data output format is used for illustration.

The output format is:

mmllll/wwww $h_1h_1h_1h_1$ $h_2h_2h_2h_2$

Where: mm is the two least significant digits of the starting sector most significant bits in hexadecimal. The two most significant digits are always 0 due to the size of a mass memory device.

llll is the starting sector least significant bits.

wwww is the starting word in hexadecimal.

h_i is the hexadecimal data.

Convert Word Address to Sector/Word Address

This statement is used to convert the mass memory address from word to sector addressing mode. The format is:

CWA,wa

The mass memory address expressed in sector addressing mode is printed on the comment unit with the format shown below:

SECTOR/WORD ADDRESS + msb lsb word

Where: msb is the most significant bit; four hexadecimal digits

lsb is the least significant bit; four hexadecimal digits

word is four hexadecimal digits

The input value of wa is a hexadecimal number with a maximum of eight digits.

Compare Core to Mass Memory

This statement is used to compare a block of core location(s) with a block of mass memory location(s). The format is:

CCM,sc,ec,ssmsb,sslsb,sw

When differences exist between the core and mass memory blocks, locations and their contents (in hexadecimal) are printed as shown:

MASS MEMORY DATA

cccc(ssss) mmllll/wwww (yyyy)

Where: cccc is the core location.

mm is the most significant bit sector of mass memory.

llll is the least significant bit sector of mass memory.

wwww is the word address within a sector.

xxxx is the contents of the core location/

yyyy is the contents of mass memory.

Compare Mass Memory to Mass Memory

This statement is similar to the last command (CCM) except that the comparison is between two mass memory areas. The format is:

CCM,lu,ssmsb,sslsb,sw,nw,nlu,nsmsb,nslsb,nsw

The output format for any differences in the compared data is shown below:

MASS MEMORY DATA

mmllll/wwww (xxxx) mmllll/wwww (yyyy)

Set 1 of mass memory data Set 2 of mass memory data

lu and nlu allow the data areas to reside on different mass memory logical units.

Search Mass Memory for Pattern

This command is used to locate data with a certain bit pattern on mass memory. The format is:

SMN,ssmsb,sslsb,sw,nw,n,m,i

If the last parameter, i (increment location count), is omitted, a value of 1 is assumed. The list output format for data is shown:

CELL CONTENT

mmmmllll/wwww (xxxx)

MASS MEMORY OPERATIONS WITH ALTERATION

Move Mass Memory

This command allows the programmer to move a block of data on mass memory from one location to another on the same or on a different device. The contents of the new location block are the same as the old one. The format is:

MMM,lu,ssmsb,sslsb,sw,esmsb,eslsb,sw,nlu,nsmsb,nslsb,
nw nsw

The two parameters, lu and nlu, are provided for any system configuration with more than one mass memory unit.

The contents of the first word of both old and new data is printed as a safety measure and for the programmer's approval prior to moving data from one block to another. This feature allows the operator to abort the request if an error is suspected.

Modify Core Image

Data within the core image on mass memory can be modified by three types of input data:

- Hexadecimal – LHC,sc,b/h_1,h_2,h_3. . . h_n
- Decimal – LIC,sc,b/i_1,i_2,i_3. . . i_n
- ASCII – LAC,sc,b/a_1,a_2,a_3. . . a_n

The new and old (data currently contained in core image) data is printed for approval by the operator before the change is made. The output format is:

```
NEW        OLD

xxxx       yyyy

 .          .
 .          .
 .          .

   VERIFY
```

From the previous printout, the operator can determine if an error exists in an input statement. If the operator is satisfied that the data is correct, OK is entered to signal that the change is correct. Any other entry causes the request to be aborted. The following message:

DB REQUEST ABORT

is printed and input is ignored.

Modify Mass-Resident Ordinal Program

Five types of input data formats are allowed as input data entry:

- Hexadecimal – LHO,ord,sc,b/h_1,h_2,h_3, . . . h_n
- Decimal – LIO,ord,sc,b/i_1,i_2,i_3, . . . i_n
- ASCII – LAO,ord,sc,b/a_1,a_2,a_3, . . . a_n
- Single precision – LSO,ord,sc,b/s_1,s_2,s_3,s_4, . . . s_n
- Double precision – LDO,ord,sc,b/d_1,d_2, . . . d_n

The ordinal number is checked against the maximum number in the system. The following statement:

DB ORDINAL OVER MAX.

is printed when it is over the maximum. New and old data are printed for approval. When verified by the operator, data is transferred to mass memory in the specified ordinal and program location. The statement:

DB ORDINAL LENGTH ZERO

is printed if the ordinal contains no program.

Modify Mass Memory

There are five commands used to modify mass memory. The command formats are shown below for the respective data types:

- Hexadecimal – LHM,ssmsb,sslsb,sw/h_1,h_2,h_3, . . . h_n
- Decimal – LIM,ssmsb,sslsb,sw/i_1,i_2,i_3, . . . i_n
- ASCII – LAM,ssmsb,sslsb,sw/a_1,a_2,a_3,a_4, . . . a_n
- Single Precision – LSM,ssmsb,sslsb,sw/s_1,s_2,s_3, . . . s_n
- Double precision – LDM,ssmsb,sslsb,sw/d_1,d_2,d_3, . . . d_n

The new and old data are printed for verification by the operator.

BREAKPOINT PROGRAM

The breakpoint program is a software checkout tool that enables the user to control and examine (in the background) the execution of a program in checkout. This program is loaded with the checkout job and achieves its purpose by means of breakpoints. Breakpoints are locations in the program in checkout before which execution is suspended and during which the operator is interrogated for control statements.

GENERAL OPERATIONS

The breakpoint program consists of a control program and a set of overlays that act on the various control statements. The control program is stored in the system library by the system initializer on LIBEDT (section 13). It is loaded with the program in checkout when the breakpoint load flag is set and an execute statement (*X or *LGO) is encountered. The load flag is set as the result of an *B statement.

Breakpoint is entered just prior to execution and announces this event by printing BP on the comment medium. At this time the breakpoint program is ready to accept the set breakpoint control statements which are described below. After the breakpoints are set, the program in checkout is executed using an *END command. When the breakpoint program is entered because a breakpoint has been reached, it prints:

BP,xxxx

Where: xxxx is the 4-digit hexadecimal address of the breakpoint.

When the breakpoint program is entered, the user can direct further operation by means of control statements in format records of 72 characters or less. These control statements are described in the sections that follow.

On receipt of a control statement, the control program reads an associated statement processor (overlay) from the

program library. The processor was stored there as an absolute record by LIBEDT.

Control statements which cannot be recognized by the breakpoint program are rejected and indicated by means of a diagnostic (FORMAT ERROR) after which the breakpoint program waits for a new statement.

Breakpoint scans and acts on each field from left to right. Therefore, if an error is detected, all fields to the left of the error have been processed and fields to the right have not.

CONTROL STATEMENTS

Set Breakpoint

This statement is used to specify locations at which breakpoints are to be set. A breakpoint may be set at any unprotected location regardless of the instruction contained there. However, breakpoints do not work in the following locations:

- The second word of a two-word instruction
- A storage location not accessed as an instruction
- A location that is modified as a result of program execution
- The RTJ- ($F4) of a system request
- Instructions with two-way reject addresses

The user is limited to a total of 15 active breakpoints. Should he specify a sixteenth, the breakpoint program responds with:

```
TOO MANY BREAKPOINTS
xxxx
FORMAT ERROR
```

Where: xxxx is the sixteenth breakpoint as specified in the control statement.

If a user specifies a breakpoint outside the bounds of unprotected memory, the breakpoint program responds with:

```
xxxx
PROTECT ERROR
```

Where: xxxx is the bad breakpoint specified.

In addition to the above, the following errors are detected:

- Nonhexadecimal characters
- Absence of commas between successive declarations
- Addresses and/or increments of more than four hexadecimal digits

In these cases, breakpoint responds with:

```
xxxx
FORMAT ERROR
```

Where: xxxx is the erroneous field as specified in the statement.

Should a user specify a location that is already a breakpoint, the breakpoint program ignores the additional declaration. The statement format is:

$*SET,b_1 + i_1,b_2 + i_2, \ldots ,b_i + i_i$

Where: b is a hexadecimal address of four characters or less.

i is a positive hexadecimal increment of four characters or less.

Either b or i may be omitted, in which case the value is assumed to be 0.

Example:

```
*SET,105A+47,,1B01,+1BA1
```

Where: 105A+47 causes a breakpoint to be set at 10A1

1B01 causes a breakpoint to be set at 1B01

,, implies omission of both b and i, and the breakpoint program would assume that the user wants a breakpoint at location 0 (which is illegal)

+1BA1 causes a breakpoint to be set at location 1BA1

Terminate Breakpoint

This statement is used to remove breakpoints previously set by the *SET command. Its format is:

$*TRM,b_1+i_1,b_1+i_2, \ldots ,b_1+i_1$

Where: b is a hexadecimal base address of four digits or less.

i is a positive hexadecimal increment of four digits or less, which, when added to its associated base, gives the actual breakpoint address.

To remove a breakpoint it is not necessary to specify it in exactly the same form as was used to set it (*SET). It is only necessary that the sums be equal. Either v or i may be omitted; omitted parameters are assumed to be 0.

The following illegalities can be detected:

- Nonhexadecimal characters
- Absence of commas between successive declarations

- Breakpoints that do not exist
- More than four hexadecimal digits in a base address or increment

In each case the breakpoint program responds with:

```
xxxx
FORMAT ERROR
```

Where: xxxx is the erroneous field as specified in this statement.

Example:

```
*TRM,10A1,+10A1,1001+A0
```

Where: 10A1 omits the increment; the first breakpoint specified in the example under *SET would be deleted.

+10A1 causes the same effect as the first field, only the base has been omitted; it would be illegal if presented as shown since the first field has already caused deletion of the breakpoint.

1001+A0 causes the same effect as 10A1; it would be illegal if presented as shown since the first field would previously have caused deletion of the breakpoint.

Entry Of Data Into Core

Data may be entered into memory in any of three forms: hexadecimal integers, decimal integers, and ASCII characters. There is a unique control statement associated with each type of data. The statements are described below.

Hexadecimal Data

The statement format is:

```
*LHX,b,i/h,h, . . . h
```

Where: b is a base address of four hexadecimal digits or less.

i is an increment of four hexadecimal digits or less.

h is a hexadecimal integer of four digits or less. There can be as many hexadecimal integers in a statement as can be accommodated on a teletype line.

The breakpoint program adds b and i together to obtain the effective storage address for the first piece of data. Data is stored in sequential locations starting at b+i. Either b or i may be omitted; when omitted, they are assumed to be 0. Hexadecimal integers that are omitted are signaled by successive commas. When an h is omitted, it is assumed to be 0.

The following errors can be detected:

- Nonhexadecimal characters in the address or data
- Delimiters other than commas, or, in one instance, a slash
- More than four digits in an integer
- An attempt to store out of bounds

Errors 1 through 3 cause the following diagnostic:

```
xxxx
FORMAT ERROR
```

Where: xxxx is the erroneous field.

Error 4 causes the following diagnostic:

```
xxxx
PROTECT ERROR
```

Where: xxxx is the data that the user attempted to store improperly.

Example:

```
*LHX,4ACF,1E8/A,,BCDE
```

This statement causes data to be stored in the location starting at 4CB7=4ACF+1E8. The data stored is A, 0, and BCDE.

Alternate forms of the statement are:

```
*LHX,4CB7,/A,,BCDE      (i omitted)
*LHX,4CB7/A,,BCDE       (b omitted)
```

Decimal Data

The statement format is:

```
*LIT,b,i/d,d, . . d
```

Where: b is a base address of four hexadecimal digits or less.

i is an increment of four hexadecimal digits or less.

d is a decimal integer of five digits or less. There can be as many decimal integers in a statement as can be accommodated on a teletype line.

Breakpoint adds b and i together to obtain the effective storage address for the first piece of data. Data is stored in sequential locations starting at b+i. Either b or i may be omitted; when omitted, they are assumed to be 0. Decimal integers that are omitted are signaled by means of successive commas with a value of 0. The following errors can be detected.

- Nonhexadecimal characters in address or increment
- Delimiters other than commas, or, in one instance, a slash
- More than four digits in a hexadecimal integer
- More than five digits in a decimal integer
- Nondecimal digits in a data field
- An attempt to store out of bounds

Errors 1 through 5 cause the following diagnostic:

```
xxxx
FORMAT ERROR
```

Where: xxxx is the erroneous field.

Error 6 causes the following diagnostic:

```
xxxx
PROTECT ERROR
```

Where: xxxx is the data that the user attempted to store improperly.

ASCII Data

The statement format is:

*LAS,b,i/aaa. . .a

Where: b and i have the same meaning and restrictions as *LHX.

a is a null character to be transmitted to memory. The ASCII characters (carriage return, line feed, cancel, and null) that are not transmitted by the driver are not stored.

Core Dumps

Core can be dumped in three data forms: hexadecimal integers, decimal integers, or ASCII characters. A separate statement exists for each data form; however, the parameters for the three statements are identical and have identical restrictions.

The three data statements are:

*DPC,s,e,b	Hexadecimal output
*DIC,s,e,b	Decimal output
*DAS,s,e,b	ASCII output

Where: s is the starting address for the dump.

e is the ending address for the dump.

b is the base address.

All parameters are hexadecimal integers of four digits or less. If a parameter is omitted, it is assumed to be 0. b is used as follows.

b+s = Actual starting address for the dump

b+e = Actual ending address for the dump

The error conditions that can be detected are:

- Hexadecimal integers of more than four digits
- Nonhexadecimal characters in a hexadecimal integer
- Noncomma delimiters
- The end address is less than the start address

Errors 1 through 3 cause the following diagnostic:

```
xxxx
FORMAT ERROR
```

Where: xxxx is the contents of the erroneous field. The error 4 diagnostic omits the field contents. Output is in nine columns: the first column contains the address of the first data item; subsequent columns contain data from sequential cells.

Example:

*DPC,,A,43A0

Where: s is omitted

e is 11

b is 43A0

The output is as follows:

```
43A0   xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
43A8   xxxx xxxx xxxx
```

Where: xxxx is a four-digit hexadecimal integer, a five-digit decimal integer, or two ASCII characters.

Mass Storage Dump

The breakpoint program is capable of dumping portions of the scratch area on the library mass storage unit. Output can be in hexadecimal, decimal, or ASCII form. There is a unique statement for each output form, though the parameters for all three statements are identical in form.

The statement forms are:

*DMH,m,l,s,n	Hexadecimal dump
*DMI,m,l,s,n	Decimal dump
*DMA,m,l,s,n	ASCII dump

Where: m is the most significant bits of the starting scratch sector number; up to four hexadecimal digits.

l is the least significant bits of the starting scratch sector number; up to four hexadecimal digits.

s is the starting word number in the starting sector; up to four hexadecimal digits between 0 and 7FFF.

n is the number of words to dump; up to four hexadecimal digits.

Any parameter that is omitted is assumed to be 0. An exception is n, which cannot be 0 and therefore cannot be omitted.

The error conditions that can be detected are:

- Nonhexadecimal digit found in a hexadecimal number
- Too many digits in a number
- Number is out of the expected range
- Noncomma delimiter

When detected, an error condition results in the following diagnostic:

```
xxxx
FORMAT ERROR
```

Where: xxxx is the erroneous field.

Output is in blocks of sector size, each block headed by its sector number. The blocks are in nine columns, of which eight are data. The leftmost (first) column contains the word number of the data in the same row of the next (second) column. Subsequent data elements in a row have sequential word numbers.

Example:

```
*DMH,,1,50,12
```

This statement requires that data be dumped from scratch sector 1 starting at word 50_{16}. 12_{16} words are to be dumped. Output would appear as follows:

```
0000  0001
0050  xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
0058  xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
0000  0002
0000  xxxx xxxx
```

Where: xxxx represents hexadecimal data.

Resume

When a user has completed work at a breakpoint, or prior to initial program execution, the *END statement is used to inform the breakpoint program. There are no parameters.

On receipt of this statement, the breakpoint program resets A, Q, and i to the latest object program values, executes the instruction at the breakpoint interpretively, and resumes object code execution. If the entry to breakpoint was before program execution, the *END statement causes the breakpoint program only to enter the object program at its entry point. There are no error conditions or diagnostics associated with this statement.

Jump

A user can cause the breakpoint program to execute a jump to any location in the background by entering the following statement:

```
*JP,b+i
```

Where: b is a hexadecimal address of four digits or less.

i is a hexadecimal value of four digits or less which, when added to b, gives the effective jump address.

Either b or i can be omitted, in which case they are assumed to be 0.

The error conditions that can be detected are:

- Nonhexadecimal characters in a number
- More than four characters in a hexadecimal number
- Illegal delimiter (+)
- An effective jump address outside the bounds of unprotected memory

The first three errors cause the following diagnostic:

```
FORMAT ERROR
```

The jump is made with A, Q, and I containing the latest object program values.

Examples:

```
*JP,2A43        (i omitted)
*JP,+2A43       (b omitted)
*JP,2A00+43     (neither omitted)
```

All three examples cause a jump to 2A43 if that location is within the bounds of unprotected memory.

Return Jump

The breakpoint program executes a return jump command from a user. The statement form is:

```
*RJ,b+i
```

Parameters and conditions are the same as for *JP, (see the Jump section). If the subroutine entered returns, the breakpoint program resumes operation at the current breakpoint.

Change Of Logical Unit

The breakpoint program initially makes use of the comment input and output devices for communication. A user may change this for the breakpoint program only by the following statements:

*LUI,1	Change input device to logical unit 1 (a decimal number).
*LUO,1	Change output device to logical unit 1 (a decimal number).

The breakpoint program checks 1 to ensure that it is a legal logical unit number and is available to the background for input or output. If this is not the case, the breakpoint program prints the following diagnostic:

FORMAT ERROR

Setting Of A,Q, Or I

When the breakpoint program is entered, it saves the current values in A, Q, and I. When the object program is resumed, the registers are restored to these values. If a user wishes to change any or all of these saved values, he may do so by means of the following statements:

*SAH,b+i	Change A.
*SQH,b+i	Change Q.
*SIH,b+i	Change I.

Where: b is a hexadecimal base of four digits or less.

i is a hexadecimal increment of four digits or less.

The value inserted into the indicated register is b+i. Either b or i or both may be omitted, in which case they are assumed to be 0.

The following errors can be detected:

- More than four hexadecimal digits in a number
- Nonhexadecimal characters in a number
- Illegal delimiter between b and i

Any of these errors cause the following diagnostic:

FORMAT ERROR

Examples:

*SQH,5	(i omitted)
*SAH,+5	(b omitted)
*SIH,3+2	
*SQH,	

The first three examples set all registers to 5. The last sets Q to 0 since all parameters have been omitted.

List Registers

The contents of A, Q, I, M, and P on entry to the breakpoint program are saved for resumption of the object program. The user can change A, Q, and I and display them by the *LRG statement. There are no parameters or illegalities associated with this statement.

The output has the following form:

P = xxxx A = xxxx Q = xxxx I = xxxx M = xxxx

Where: xxxx is the hexadecimal contents of the associated register.

Motion Request

Breakpoint performs motion operations via the commands described below. In each case the logical unit specified must be available to background programs.

Advance File

The advance file statement format is:

*ADF,1,n

Where: 1 is the logical unit (a decimal number).

n is the number of files to skip (a decimal number).

If n is omitted it is assumed to be 1. The following errors can be detected:

- Illegal logical unit number
- Nondecimal character in a decimal number
- A decimal number greater than five digits
- Illegal delimiter

In all cases the diagnostic is:

FORMAT ERROR

Example:

*ADF,5

If logical unit 5 is available to the background, it is moved forward until the first file mark is sensed.

Backspace File

The backspace file format is:

*BSF,1,n

Parameters and conditions are the same as for *ADF. *BSF causes the tape to be searched for file marks in a reverse direction.

Advance Record

The advance record statement format is:

*ADR,1,n

The parameters and conditions are the same as for *ADF, except that the tape searches forward for end-of-record gaps.

Backspace record

The backspace record statement format is:

*BSR,1,n

The parameters and conditions are the same as for *ADF. Operation is the same as *BSF, except that the tape searches in reverse for end-of-record gaps.

Write End-of-File

The write end-of-file statement format is:

*WEF,1,n

The parameters and conditions are the same as for *ADF. This statement causes n file marks to be written on the indicated logical unit.

Rewind

The rewind statement format is:

*REW,1

Parameter 1 and its conditions are as described under *ADF. This statement causes the indicated unit to be rewound to load point.

Unload

The unload statement format is:

*UNL,1

Parameter 1 and its conditions are as defined under *ADF. This statement causes the specified logical unit to be rewound to load point and, if possible, unloaded.

Select Density

The select density statement format is:

*SLD,1,d

Where: 1 is the logical unit number (decimal).

d is the density indicator.

0	200 bits per inch
1	556 bits per inch
2	800 bits per inch
3	1600 bits per inch
Omitted	556 bits per inch

This statement causes the breakpoint program to attempt to set the specified logical unit to the desired density. No check is made to determine if the unit is capable of the density requested.

RECOVERY PROGRAM

The programmer may, with the recovery program, determine the state of core and mass storage at the end-of-job execution. The recovery program is operated under the direction of the job processor. Whenever the job processor detects an *SR statement before job execution, it sets the RECOVERY indicator switch. The recovery program is operated after job execution when the switch is set.

The recovery program is entered after it is loaded into protected core. Entrance is indicated by the printout RE. The operator may then direct operation by control statements.

The recovery program rejects any input statement for which there is no entry in its list of control statements. Unacceptable control statements are indicated by a printout and the recovery program waits for another control statement.

CONTROL STATEMENTS

Control statements to the recovery program are formatted records. All control statements must be entered from the comment device.

The first character of a control statement must be an asterisk; the last character must be a carriage return. Intervening characters identify the type of statement and action.

Unacceptable control statements are indicated by the printout ERR, and the recovery program waits for another control statement.

The following are some of the standard input statements for the recovery program:

Input Statement	Transfer Address
*Dssss,eeee	DMPCOR
*Ms_2,w_1,s_2,w_2,n	MASDMP
*T	TERMIN
*unit number	OUTSEL

Dump Core

The contents of core are dumped beginning at ssss and terminating at eeee.

The format is:

*Dssss,eeee

Where: ssss is the starting address in hexadecimal digits. If eeee is less than ssss, no data is printed. If eeee is omitted, only one word is printed in hexadecimal digits.

eeee is the ending absolute address in hexadecimal digits.

Mass Storage Dump

Any area on mass storage can be dumped (not just the scratch area) beginning at word 1 in sector 1 and terminating at word 2 in sector 2.

The format is:

*Ms$_1$,w$_1$,s$_2$,w$_2$,n

Where: s$_1$ is actual sector 1. If s$_1$ is omitted, dumping begins with the first sector on the mass storage device.

If s$_1$ is the only value specified, a single sector is dumped.

If s$_1$ is greater than s$_2$, no information is dumped.

If s$_1$ is equal to s$_2$ and w$_1$ is greater than w$_2$, no information is dumped.

w$_1$ is word 1. If w$_1$ is omitted, dumping begins with the first word of the first sector.

s$_2$ is sector 2. If s$_2$ is omitted, s$_2$ has the same value as s$_1$.

w$_2$ is word 2. If w$_2$ is omitted, the last word to be dumped is the last word in sector 2.

n is the logical unit number of the mass storage device. If omitted, the mass storage unit containing the library is assumed.

Terminate

This statement terminates recovery program control and the job processor is re-entered.

The format is:

*T

Select Output Device

This statement allows the operator to select a device for output by the recovery program. The unit number must be legal and the device must be capable of output. If this statement is not used, the standard print output device is used.

The format is:

*unit number

ADDITION OF CONTROL STATEMENTS

Control statements may be added to the recovery program in the following manner:

1. Add the statement and external name (transfer address) to the list.
2. Supply a function module, beginning at the corresponding entry point, to accommodate the action required by the control statement. This function module should be processed with the control module at system initialization.

SYSTEM ABORT DUMP

When a system condition causes abnormal termination of system operation, the core-resident system abort dump program can be executed to dump the contents of selected core locations. There is a unique dump routine for each type of line printer (DMP421 for the 1742-1 Line Printer, DMP42x for the 1742-30/1742-120 Line Printer, and DMP827 for the 1827-30/65119-1 Line Printer). The procedure to operate the program is:

1. Press MASTER CLEAR.
2. Enter the starting address of the location to be dumped in the A register.
3. Enter the ending address of the location to be dumped in the Q register.
4. Set the P register to the address 140_{16}.
5. Execute the program by setting the RUN/STEP switch to RUN.

The following results after the program is executed:

1. The paper is set to the top of the form.
2. There is an absolute/relative heading of 16 columns at the top of each page.
3. Absolute and relative addresses and 16 words are printed per printer line.
4. Lines, whose 16 words are the same as the last line printed, are ignored by printing a line of asterisks.
5. Sixty lines are printed per page.
6. The program hangs when the requested number of words is printed.

The system abort dump program can be executed as many times as is required to dump the selected contents of core by repeating the operating procedure.

The printout is ishown in figure 10-1.

ABSL	REL	50 †	61	72	83	94	A5	B6	C7	D8	E9	FA	0B	1C	2D	3E	4F
1F75	1A70	1CF7	0000	0000	0000	0000	0180	0B00	08FB	48FB	E105	0FA6	0D03	0A0F	5400	1797	C8F3
1F85	1A80	E8F3	0F61	48EF	0121	184F	C800	FC67	09FC	0100	D800	FC63	5802	182D	0D00	5800	FD79
1F95	1A90	0FC7	0132	08E2	0337	1CF8	C800	FC58	09FD	0111	1CDC	D800	FC53	0A00	68D3	58EE	5800
1FA5	1AA0	FD0R	18FD	05CE	AC3D	011F	58E7	5800	EDD4	18FD	A02D	0119	D8C5	58E0	5800	FCFD	1800
1FB5	1AB0	FFFC	A02D	0111	1804	C8DC	09FE	688A	8D9	0113	E888	0FA1	1A12	09FE	68D3	5800	FD49
1FC5	1AC0	C02E	0309	0DFE	0202	18FE	0B00	5800	ECE4	18FA	18ED	0B00	1800	FCAE	1800	FF1C	1800
1FD5	1AD0	FD43	1800	FE15	0800	FC19	09FD	0101	8B1	C80C	FC15	D800	FC13	09FD	0138	0111	1820
1FE5	1AE0	0CC0	4800	FC0B	4300	FC0A	1C90	58A6	E110	0FAA	C02C	0172	C000	022C	2000	000C	6813
1FF5	1AF0	E106	54EF	0DFE	0F6F	380E	680D	5800	FD11	C80B	03DC	5892	C807	0102	09FE	18F6	6800

Figure 10-1. System Abort Dump Program Printout

NOTE

In order to conserve core memory, the train image is not loaded when the 1742-120 Line Printer is used (assuming an existing train image).

CYBER 18 EXTENDED MEMORY ABOUT DUMP

A CYBER 18 with extended memory requires a different dump routine (ECMDMP) if the capability to dump the extended memory is required.

The procedure to operate the program is:

1. Press MASTER CLEAR.
2. Depress ESCAPE.
3. Enter K31000800G.
4. Enter the starting address of the location to be dumped in the A register.
5. Set the ending address of the locations to be dumped in the Q register.
6. Set the M register with a value for the 65 K bank from which the dump will occur.

   m=0 Dump the page file values and the current locations specified by the page files.

   m=1 Dump from the first 65 K physical bank.

   m=2 Dump from the second 65 K physical bank.
7. Set the P reister to the address 140_{16}.
8. Execute the program by setting RUN mode. The line printer must be ready.

The following results after the program is executed:

1. The paper is set to top-of-form.
2. If M is zero, the 32-page mode 0 registers are printed, a top-of-form is done, the 32-page mode 1 registers are printed, and a top-of-form is done.
3. There is an absolute relative heading of 16 columns at the top of each page.
4. The physical address upper, the physical address lower, the logical address, the relative address, and the contents of 16 memory locations are printed.
5. Lines, whose 16 words are the same as the last line printed, are ignored by printing a line of asterisks.
6. The program hangs when the requested number of words is printed.

The system abort dump program can be executed as many times as is required to dump the selected contents of core by repeating the operating procedure.

The printout is shown in figure 10-2.

ON-LINE SNAP DUMP

This routine dumps the contents of the A, Q, M, and I registers, along with the contents of P, which equals the next program address to be executed after the call to dump. The register contents are temporarily saved in a circular-managed buffer to allow multiple use of the dump routine. The dump information is converted to print format and output at the output device speed. If multiple calls to the dump routine fill the available buffer area (standard set-up allows five dumps to be stacked), future dump calls are ignored until a previous entry output is completed, making entry into the buffer possible.

This routine is an optional debugging aid that may be deleted from the system. If included, the routine is resident in protected core. Any user program may call the dump routine. Unprotected program communication is provided by an entry in the table of PRESETS.

The calling sequence for an on-line dump is a return jump from the user program to the entry point SNAPOL. Any user program with a calling sequence to the dump routine must declare SNAPOL as an external. Output is directed to the standard list device on a FIFO basis in the following format:

P = pppp Q = qqqq A = aaaa M = mmmm I = iiii

Control returns to the location following the calling sequence (return jump) with the program state the same as it was before the call for a dump.

```
                                  90†  A1   B2   C3   D4   E5   F6   07   18   29   3A   4B   5C   6D   7E   8F
0000  1579  1579  0000          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
0000  1449  1449  0050          0000 0000 14F4 6070 0002 D5FF DF49 8900 14F4 7FFF 7FFF 3131 3131 3131 0008 0008
0000  1449  1449  0050          0008 0004 0008 0003 0020 D457 029F 0005 0000 0510 0510 051F 0041 050A 0529 0713
0000  1479  1479  0100          6700 0000 0001 0000 0002 0004 0003 FFFF 0004 4A50 5245 544E C40D 534F 4150 4F4C
0000  1449  1449  0110          C429 4445 4544 494C C70D 5245 4C44 494C C711 4445 4449 445A C715 4C4F 4D4A 494C
```

†The 9 is absolute; the 0 is relative.

Figure 10-2. Extended Memory Abort Dump Program Printout

System checkout is an on-line program that diagnoses failures in a CYBER 18/1700 Mass Storage Operating System (MSOS).

The failed system image is written on mass storage by one of the following bootstrap programs:

- B1751 – 1751 Drum
- B1752 – 1752 Drum
- BDK85X – { 1733-1/853/854 Disk / 1738/853/854 Disk }
- B17332 – 1733-2/856-2/856-4 Cartridge Disk
- B17391 – 1739-1 Cartridge Disk
- B18331 – 1833-1/1833-3/1867-10/1867-20 Storage Module Drive
- B18334 – 1833-4/1866-12/1866-14 Cartridge Disk

This bootstrap operation is followed by a system restart (autoload) and call-up of the system checkout program (SYSCOP) via MIPRO. This program executes at a low priority level, obtaining its information from the image on mass storage in an attempt to isolate the system problems.

System checkout is written as a series of overlays to minimize core requirements.

CHECKOUT BOOTSTRAP PROGRAMS

The checkout bootstrap programs write the core image on mass storage; they are self-contained and thus require no drivers.

ASSUMPTIONS AND RESTRICTIONS

The user is responsible for ensuring that the bootstrap program is intact and in core. The bootstrap must be core-resident, as it is referenced absolutely by the user.

The bootstrap program transfers the number of words specified by the user via SYSDAT to mass storage.

The starting sector number is specified by an EQU in SYSDAT.

When using a cartridge disk, the failed image must reside completely on either disk 0 or disk 1.

When using the storage module drive, the failed image must reside completely within the lower $7FFF_{16}$ sectors.

The release system has standard values specified for memory size and the mass memory location of the failed image. Refer to the MSOS Customization Manual for procedures to alter these parameters. The system's A, Q, and I registers are saved by the bootstrap program for reference by SYSCOP.

COMPLETION AND ERRORS

After transferring the failed image to mass storage, the bootstrap stops (loops on a selective stop) with the Q register set to 0 if no errors occurred during the transfer or negative if errors occurred.

BOOTSTRAP OPERATION

When the system fails, the following steps are used to bootstrap the failed system onto mass memory:

1. Stop the computer. Do not press MASTER CLEAR.
2. Clear the M, P, Y, and X registers.
3. Set the P register to the address 142_{16}.
4. Set the SELECTIVE STOP switch. Select the Q register.
5. Place the computer in RUN. The computer stops when the failed image has been transferred. If Q is zero, go to step 6. Otherwise, a transfer error has occurred and re-try from step 2.
6. Autoload the system.
7. After system start-up, request SYSCOP via MIPRO.

SYSTEM CHECKOUT PROGRAM

The system checkout program (SYSCOP) systematically examines the failed image for evidence of failures in the CYBER 18/1700 MSOS. When an error is encountered or questionable information is found, an appropriate message is given.

STRUCTURE

SYSCOP is divided into a control module and segment portions. The control module remains in core while the checkout program is running. It reads in each segment through a mass memory request as it is required. The control module includes subroutines common to all segments. SYSCOP segments are structures to conform to the 96-word sectors.

The program executes in a series of overlays, thus minimizing core requirements.

MESSAGES

The system checkout program produces three categories of messages: control, error, and support. The operator selects the type of message. All numbers included in the messages are given in hexadecimal.

CONTROL MESSAGES

The system checkout program gives messages to control the operation of SYSCOP. Control messages appear on the list device unless operator intervention is required. In this case the control message and its associated input are via the comment device.

Control messages always appear, regardless of the message option selected.

ERROR MESSAGES

Error messages indicate that an error condition was detected. Gross error detection messages, as well as specific error messages, are included in this level of messages.

Error messages appear on the list device.

SUPPORT MESSAGES

The system checkout program uses support messages to support, expand, and present information to the user. Support messages supply the user with organized information that may help in isolating errors.

Support messages may not always be related to an error.

All support messages appear on the list device.

ERROR CHECKS

The following sections describe the various checks performed by the system checkout program.

Set-Up

The set-up routine initializes the system checkout program, informs the operator of the program's presence, gives the sector number for the beginning of the failed system image, and allows the operator to select a message option.

Initial Message

The following control message indicates the start of the system checkout program:

SYSCOP START

Acknowledgement of Image Sector

The following control message acknowledges the beginning of the image sector:

IMAGE START SECTOR IS ssss

Where: ssss is the starting sector of the failed image.

Selecting the Message Option

The following control message indicates operator selection of the message option:

SELECT OPTION

s (reponse)

Where: s is the desired option.

*Z	Checkout package is released.
0	Control is transferred to the dump routine.
1	Output error messages only
2	Output error messages and support messages associated with detected errors
3	Output error messages and all support messages

When 1, 2, or 3 is completed, the user is again asked to select options. After a dump is completed, the typeout DUMP is repeated. The user may then return to select options by entering an *R, executing another dump, or releasing SYSCOP.

An undefined option or an error on the input comment device causes SYSCOP to re-issue the option message.

Register Contents

This segment of the system checkout program prints the contents of the registers as saved by the checkout bootstrap program.

This support message appears as follows:

A	Q	I	REGISTER
aaaa	qqqq	iiii	

Where: aaaa is the contents of the A register.

qqqq is the contents of the Q register.

iiii is the contents of the I register.

Locore Constants

The system checkout program analyzes the various constants, both numbers and addresses, contained in the communications region, for possible errors. Both the failed image and the autoload image are checked.

The following error message indicates an error detected on the autoload image:

***LOCORE CONSTANT ERROR INITIALLY

The messages that follow refer to these errors. If no error is found on the autoload image, the message does not appear.

The following error message indicates an error detected on the failed image:

***LOCORE CONSTANT ERROR

The succeeding messages refer to those errors. If no error exists on the failed image, the message is suppressed.

LOCORE Bit Table Error

The following error message indicates an incorrect checksum of the total of locations 2_{16} through 46_{16}:

BIT TABLE CHECKSUM ERROR

This indicates that at least one location between 2_{16} and 46_{16} inclusive has been altered. If no error is detected, the message does not appear.

LOCORE Communication Address Error

The following error message appears each time an altered address is found in LOCORE:

ADDRESS IN aa WAS ffff BUT SHOULD BE iiii

Where: aa is the address of the LOCORE location containing a monitor address.

ffff is the value at the time of failure.

iiii is the value at the time of initialization.

The list of addresses checked for alteration include:

$B5_{16}$	FNR
$B6_{16}$	COMPRQ
$B7_{16}$	MASKT
$B9_{16}$	REQST
BA_{16}	VOLR
BB_{16}	VOLA
BC_{16}	LUABS
BD_{16}	SABS
BE_{16}	CABS
BF_{16}	NABS
$E9_{16}$	EXTBV4
EA_{16}	DISPxx
$F4_{16}$	MONI
$F8_{16}$	IPROC
FE_{16}	ALLIN

Error in Core Bounds

The following error message indicates that the unprotected bounds exceed the limits of core, the top of unprotected is below the bottom, or any of the addresses is negative if in 32K mode:

MAX CORE WAS hhhh WITH iiii TO jjjj UNPROT (ERROR)

Where: hhhh is the contents of $F5_{16}$.

iiii is the contents of $F7_{16} + 1$.

jjjj is the contents of $F6_{16} - 1$.

Highest Core Location and Bounds of Unprotected Core

The following support message indicates that no location error was detected. This message appears only if option 2 or 3 is selected.

MAX CORE WAS hhhh WITH iiii TO jjjj UNPROT

Where: hhhh is the contents of $F5_{16}$.

iiii is the contents of $F7_{16} + 1$.

jjjj is the contents of $F6_{16} - 1$.

Error in MAXSEC

The following error message indicates that the most significant bits specified in MAXSEC were nonzero:

MAXSEC WAS hhhhhhhh (ERROR)

Where: hhhhhhhh is the most significant bits (MSB).

MAXSEC Value

The support message for the error in MAXSEC is:

MAXSEC WAS hhhhhhhh

The MAXSEC and MAXCORE messages appear twice on the printout. The first is for the autoload image and the second for the failed image.

System Priority Level

The following messages analyze the system priority level. In some cases, system programs may be locked out because of a hang-up at a higher level. If the system consistently stops at a particular level, that level should be suspected. The following error message appears only if this level is two or higher:

***POSSIBLE LEVEL HANGUP

Incorrect Priority Level

The following error message indicates an incorrect priority level (not between -1 and 15):

PRI LVL WAS hhhh (ERROR)

Where: hhhh is the priority level of the system at the time the image was written on mass storage; the value is from EF_{16}.

System Priority Level

The following support message gives the system priority level and is printed only to aid subsequent debugging:

PRI LVL WAS hhhh

Where: hhhh is the priority level of the system.

Interrupt Trap Region

This section of messages detects possible errors in the interrupt trap region.

The following error message is detected on the autoload image in the trap region:

***INTERRUPT TRAP ERROR INITIALLY

The following error message indicates an error in the failed image:

***INTERRUPT TRAP ERROR

The succeeding messages aid in isolating an error in the interrupt trap region.

Unpatched Interrupt Response Routines

The following error message indicates unpatched interrupt response routines:

LINE ii RESPONSE IS UNPATCHED

Where: ii is a hexadecimal interrupt line number that had a $7FFF_{16}$ (unpatched external) for the address of its interrupt processing routine.

Line 0 Error

The following error message indicates a line 0 error. The priority level for line 0 is assumed to be F_{16} and the response routine is the Internal Interrupt Handler.

LINE 0 IS NOT SETUP FOR PARITY/PROTECT

Interrupt on an Invalid Line

The following error message indicates an interrupt on an invalid line. The specified line has INVINT as its response routine, yet an interrupt has occurred.

LINE ii LAST INTERRUPTED tttt (INVALID)

Where: ii is the line number.

tttt is the location specified in the appropriate trap.

Last Location Interrupted by Each Valid Line

The following support message indicates that an interrupt occurred on a line. The line 1 trap is also used by the monitor to initiate or resume a program's operations.

LINE ii LAST INTERRUPTED tttt

Where: ii is the line number.

tttt is the location specified in the appropriate trap.

Line vs. Level Printout

The following support message gives the line and level status:

```
LINE   0 1 2 3 4 5 6 7 8 9 A B C D E F
LEVEL  h h h h h h h h h h h h h h h h
```

Where: h is the level indicated in the trap region.

Interrupt Set for a FORTRAN Level

The following error message indicates that the interrupts cannot use the level reserved for FORTRAN. When FMASK is unpatched (7FFF), it is assumed that no FORTRAN levels are reserved.

LEVEL hh IS USED FOR INTERRUPTS AND
IS RESERVED FOR FORTRAN

Where: hh is the priority level number.

FORTRAN Levels Error

The following error message notes errors in the FORTRAN levels. There are five priority levels between 3 and E.

```
FORTRAN LEVELS
h i j k l (ERROR)
```

Where: h, i, j, k, and l are priority level numbers.

FORTRAN Levels

The following support message designates the legal levels reserved as FORTRAN levels in FMASK:

```
FORTRAN LEVELS
h i j k l
```

Where: h, i, j, k, and l are priority level numbers.

Inconsistent Use of RDISP or NDISP and FORTRAN Levels

The following error message appears if more than one FORTRAN level was reserved in FMASK, but the system is using NDISP instead of RDISP:

```
SYSTEM USING NDISP WITH REENT FORTRAN
(ERROR)
```

Interrupt Stack And Mask Table

This section of messages lists errors in the interrupt stack and mask table.

The following error message verifies that an error was detected in the autoload image mask table:

```
***MASK TABLE ERROR INITIALLY
```

The following error message indicates that the failed image mask table contains an error or has been modified:

```
***MASK TABLE ERROR
```

The succeeding messages indicate the type of error within the interrupt stack and mask table.

Mask Table Error

The following error message appears each time a bit is encountered in the mask table for a line at a higher level than the level indicated in the trap region.

```
LINE hh IS SET FOR LVL iiii BUT UNABLE TO
INTERRUPT jjjj
```

Where: hh is the line number.

iiii and jjjj are priority level numbers; iiii is a lower level than jjjj.

A similar error message appears when no bit is detected in the mask tables for lower level masks:

```
LINE hh IS SET FOR LVL jjjj BUT IS ABLE TO
INTERRUPT iiii
```

Where: hh is the line number.

iiii and jjjj are priority level numbers; iiii is a lower level than jjjj.

Mask Table Altered

The following support message gives the image for each level entry in the mask table that was modified:

```
ENTRY FOR LVL hhhh INITIALLY iiii CHANGED TO jjjj
```

Where: hhhh is the level of the mask table entry -1 to F.

iiii is the value on the autoload image.

jjjj is the value on the failed image.

Interrupt Stack Entries

The following support message gives the interrupt stack entries:

```
INTRPT STACK LEVEL
h i j k l m n o p q r s t u v w
```

Where: h through w are levels of the entries in the interrupt stack. The lowest level is h and should be -1; the highest permissible level is E. The maximum number of entries is 16.

If any of these conditions is violated or levels are not in ascending order, an error has occurred. A level can appear only once; nothing appears if the stack is empty and the priority level was -1.

Logical Unit Capability

This section of messages checks the read/write capability of the standard logical unit assignment and its alternates.

The following error message indicates that the autoload image has logical units with illegal capabilities:

```
***LOGICAL UNIT CAPABILITY ERROR INITIALLY
```

The following error message indicates that the failed image is incorrect:

```
***LOGICAL UNIT CAPABILITY ERROR
```

The next two messages further explain logical unit errors.

Standard I/O Logical Units Read/Write Capability Error

The following error message appears for each input device that cannot be read or for each output device that cannot write:

aaa IS NOT A bbbb DEVICE

Where: aaa is one of the following devices:

SBI	Standard binary input device specified in $F9_{16}$
SBO	Standard binary output device specified in FA_{16}
SLO	Standard print output specified in FB_{16}
SCO	Output comment specified in FC_{16}
SCI	Input comment specified in FD_{16}

bbbb is either READ or WRIT.

Alternate Logical Units Capability Error

The following error message indicates that the alternate device does not have the read/write capability specified for the primary device:

LU aa IS ALTERNATE FOR lu, BUT HAS LESS CAPABILITY

Where: aa is the assigned alternate logical unit for logical unit lu.

lu is the logical unit number.

Logical Unit And Physical Device Table

This section of messages analyzes the logical unit tables and the physical device table. An investigation of active I/O requests is also included.

The following error message indicates an error was detected on the autoload image in the logical unit tables:

***LOGICAL UNIT TABLE ERROR INITIALLY

The following error message indicates an error was detected on the failed image:

***LOGICAL UNIT TABLE ERROR

The succeeding messages aid in diagnosing errors in the logical unit tables and the physical device table.

Number of Logical Units

The following error message indicates that LOG1A, LOG1, and LOG2 do not contain the same number of logical units. This message does not appear if the first word of each of the three tables agrees.

NUM OF LUS DO NOT AGREE, ASSUME lu

Where: lu is the number of logical units specified in LOG1A.

Logical Unit 1 Must Be Core Allocator

The following error message indicates the equipment type code, if logical unit 1 does not specify the software core allocator. If logical unit 1 is the core allocator, the message is suppressed.

LU 1 NOT CORE ALLOCATOR

Physical Device Table Setup

The following error message indicates that the particular LOG1A entry does not point to a core location that contains a scheduler request code ($52xx_{16}$) followed by three cells (none of which is unpatched). This message appears for each error:

NO VALID PHYSTB FOR LU lu

Where: lu is the logical unit number.

Marked Down Logical Units

The following support message indicates that bit 13 of the LOG1 table reflects an inoperative logical unit. This message appears for each logical unit marked down:

LU lu WAS MARKED DOWN

Where: lu is the logical unit number.

Inconsistent Shared Devices

The following error messages indicate inconsistently shared devices:

LU lu IS SHARED BUT UNMATCHED

Where: lu is the logical unit in which bit 14 of the LOG1 table entry is set, but for which there is no other logical unit with an identical physical device table in LOG1A.

LU lu AND vv MATCH BUT SHARED BIT IS NOT SET

Where: lu and vv are logical units whose physical device table addresses match in LOG1A, but the LOG1 entry for logical unit lu does not indicate a shared device.

Analysis for Active Drivers

The following support message appears for each busy device. A device is considered busy if a nonzero logical unit appears in word 5 of the physical device table.

```
LU lu CURRENT PARA LIST AT iiii
RC    jjjj        Request code
C     kkkk        Completion address
TH    llll        Thread
LU    mmmm        Logical unit
N     nnnn        Number of words to transfer
S     oooo        Starting address
I/O IN PROGRESS
```

Where: lu is the active logical unit.

iiii is the parameter list address contained in word 6 of the driver's physical device tables; specifies the last parameter list that the driver operated on.

jjjj through oooo is the hexadecimal dump of the parameter list at location iiii.

The last line of this support message does not appear if the diagnostic clock (word 4) is set minus (device idle).

Logical Unit Threads

The next two messages give information about the logical unit threads. The first support message lists the elements of the thread until it encounters an empty entry ($FFFF_{16}$):

```
LU lu THREAD
jjjj kkkk llll mmmm nnnn oooo pppp qqqq rrrr....
```

Where: lu is the logical unit whose LOG2 entry is not $FFFF_{16}$.

jjjj. . . are entries on the thread.

If more than 40_{16} elements are on the thread, only the first 40_{16} are listed and the following message is given:

LU lu THREAD MAY BE BROKEN

Last Return Addresses for FNR and NCMPRQ

The following support message gives the last return addresses for FNR and NCMPRQ:

```
RETURN FOR FNR WAS hhhh
RETURN FOR CMR WAS iiii
```

Where: hhhh is the last location to call find next request; it should be in a driver.

iiii is the last location to call complete request; it should be in a driver.

Scheduler Stack and Volatile Storage

The following error message indicates that the levels in the scheduler stack are inconsistent. The priority level at the time of failure is also checked.

***SCHEDULER STACK ERROR

The succeeding messages further explain possible errors on the scheduler stack and volatile storage.

Scheduler Stack Entries

The next three support messages give information concerning the scheduler stack entries:

```
NUM OF SCHEDL STACK ENTRIES WAS hh
NUM OF SCHEDL CALLS STACKED WAS ii
```

Where: hh is the total number of scheduler entries defined in the system.

ii is the number of scheduler entries that were queued when the system failed.

```
SCHEDL STACK ENTRIES
hhhh/iiii jjjj kkkk llll        (A line appears for each entry)
mmmm/ . . .
```

Where: hhhh and mmmm are the addresses of scheduler stack entries.

iiii through llll is the dump of the hhhh entry.

The following message defines the last entry that was scheduled. If jjjj is zero, the message is suppressed.

```
LAST ENTRY TO BE SCHEDULED
hhhh/iiii jjjj kkkk llll
```

Volatile Storage

The following support message specifies the amount of volatile storage in use at the time of system failure:

THERE WERE hhhh OF THE iiii VOLATILE WORDS ASSIGNED

Where: hhhh is the amount of volatile storage assigned at failure.

iiii is the total volatile storage available.

Allocatable Core and System Directory

The following error message indicates that not all of allocatable core can be accounted for and a thread has been broken:

***ALLOCATABLE CORE ERROR

Allocatable Core

The following support message appears only if the block was assigned at failure. If the block was not assigned, EMPY is listed instead of an ordinal.

```
ALLOCATABLE CORE MAP
INDEX START LNGTH THRD DUMP
jjjj hhhh iiii tttt mmmm nnnn oooo pppp qqqq
EMPY kkkk llll tttt mmmm nnnn oooo pppp qqqq
```

Where:

jjjj	is the ordinal of the mass storage program in the system whose length matches the length of the block.
hhhh and kkkk	are the starting addresses of blocks of allocatable core.
iiii and llll	are lengths of block.
mmmm through qqqq	is the dump of the first five words of the block.
tttt	is the thread to the next empty block or next word.

If the length does not match the length in a directory entry, XXXX appears on the listing. A line appears for each block of core. Only the last system directory entry with a matching length appears. The length and start include the two control words preceding each block of core.

System Directory

The following error message is printed if the system directory is not set up correctly:

***SYSTEM DIRECTORY ERROR

The next two error messages give information about the request priority.

INDEX hhhh HAS INVALID REQ PRI iiii

Where:

hhhh	is the ordinal in the system directory.
iiii	is the request priority level.

The preceding message is printed for allocatable core programs.

The job processor is the only program that is permitted to have a request priority below 3. Ordinals for these modules are verified and all other programs must be at a request priority level of 4 or above. For each ordinal that does not have a valid request level, the preceding message appears.

For each system directory program that is longer than the core reserved for its request priority level, the following error message appears:

INDEX hhhh TOO LONG FOR REQ PRI iiii

Where:

hhhh	is the system directory ordinal.
iiii	is the request priority level.

Swapping and Job Processor

The system is normally swapped unless the job processing executive is operating. During the job processor's execution, swaps may occur up to some predetermined frequency. Each time a swap occurs, a level 2 idle loop is executed, which locks out all job processor execution for the duration of the swap. Thus, if the system is swapped, it should not be executing below level 2. If the system is waiting to swap, either swapping is occurring at a rate greater than the predetermined interval or the job processing executive has I/O waiting to be completed.

Swapping and Job Processor Diagnostics

The following error message indicates that the system is waiting to swap and that unprotected I/O is active:

CONSIDER UNPROTECTED I/O HANGUP

The following error message indicates that the system was kept from swapping because a set time interval had not elapsed:

CONSIDER SWAP RATE TOO RAPID

The following error message indicates that the job processor was in core and the system was swapped. This is not an error and normally occurs during job processing.

CORE USAGE CAUSED SWAP WHILE JP IN

Swap Status

The next three support messages give information concerning the swap status.

The first support message appears if the SWAPON flag was set, indicating that a swap was in effect. This flag is in the DRCORE program.

SYSTEM WAS SWAPPED

The second support message indicates the SWAPON flag and the swap waiting flag (SPASW) were not set. SPASW is in the TRVEC program.

SYSTEM NOT SWAPPED

The next support message appears if SWAPON is not set but SPASW is set.

SYSTEM NOT SWAPPED BUT WAITING TO SWAP

Job Processor in Core at Failure

The next two support messages give information about the job processor in core at the time of failure.

The first support message indicates that the job processing executive was not in core at the time of system failure. Specifically, address pointer FILE1 in the TRVEC program had a pointer of zero. No further job processor checks are made. The job processing executive maintains four files that can be located from addresses in FILE1, FILE2, FILE3, and FILE4.

JP NOT IN CORE

The second support message indicates that FILE1 contained a file address. The remainder of the job processor checks are made.

JP WAS IN CORE

Job Processor File Locations

The following support message gives the job processor file locations. If an address is zero, it implies that the respective module was not active.

FILE1	FILE2	FILE3	FILE4	LOADR	BP
hhhh	iiii	jjjj	kkkk	llll	mmmm

Where: hhhh, iiii, jjjj, and kkkk are the absolute starting addresses of the four job processor files.

llll is the starting address of the relocatable binary loader (in TRVEC).

mmmm is the starting address of the breakpoint package ($F3_{16}$).

Job Processor Lockout Switch Status

The following support message gives the job processor lockout switch status. If SWITCH in TRVEC is negative, only the first line appears. If positive, only the second line appears. This indicates the job processor is either locked out or LIBEDT or the Recovery program has requested a sign-off. If SWITCH is zero, the message does not appear.

JP LOCKED OUT FOR LIBEDT OR RECOVERY

SIGN OFF REQUESTED OF LIBEDT OR RECOVERY

Unprotected I/O and Timer Request Status

The following support message gives the unprotected I/O and timer request status. If no I/O or timer requests are active, the message does not appear.

hhhh UNPROT REQ WERE ACTIVE AND STACKED AT LOC iiii

Where: hhhh is the sum of UNPIO and UNPTIM in TRVEC.

iiii is the absolute location of the stacked requests in the protect processor (PROTEC).

Manual Interrupt Handling

The next two messages give information about manual interrupt handling. The first support message indicates the MIB flag was set and input is for the job processor.

PENDING INPUT REQUEST FOR JP

The second support message indicates that the MIB flag was set and the input is for the MIPRO program.

PENDING INPUT REQUEST FOR MIPRO

Core Dump Request

This module of the SYSCOP program executes last and permits the operator to dump selected core locations for the failed image. The valid control characters are:

*D	Output on the print logical unit.
*R	Repeat SYSCOP package with options set.
*Z	Terminate SYSCOP.

The following message indicates that the package is awaiting valid dump addresses.

DUMP

It appears after completing a request or after an invalid entry. The dump is 16 locations per line unless the comment logical unit is used, in which case the dump is eight locations per line.

The following message is the last message from SYSCOP:

FINISH SYSCOP

Partition Core

Support Messages

The following support message appears for every assigned partition:

PARTITION CORE ADDRESSES

PARTITION xx hhhh

Where: xx is partition number 0 through 15.

hhhh is the address.

The following message appears with a printout of partition and thread for every busy partition:

```
PARTITION THREADS
```

When the USE table is analyzed, each partition in use is printed with the following support message:

```
PARTITION IN USE
```

Error Messages

When an error is detected, an error header message appears:

```
PARTITION CORE ERROR
```

Partition 0 must begin at 8000_{16} or below. The following message appears when addresses above 8000_{16} are used:

```
PARTITION 0 ABOVE 8000
```

Partitions must be specified in ascending order. If this is not done, the following message appears:

```
PARTITION OUT OF ORDER
```

A bit in the busy word must be set for each permanently busy or unused partition; otherwise, the following message appears:

```
ILLEGAL BUSY INDICATOR
```

Sample Requests

Teletypewriter Listing

```
MI
SYSCOP
SELECT OPTION
0
DUMP
*D0,FF
DUMP
*R
SELECT OPTION
1
SELECT OPTION
2
SELECT OPTION
3
SELECT OPTION
*Z
MI
.
.
.
.
```

Line Printer Listing

```
SYSCOP START
IMAGE START SECTOR IS 2000
```

Dump

See figure 11-1 for a sample dump.

Option 1

```
***INTERRUPT TRAP ERROR
LINE 0A RESPONSE IS UNPATCHED
LINE 0B RESPONSE IS UNPATCHED
***INTERRUPT TRAP ERROR INITIALLY
LINE 0A RESPONSE IS UNPATCHED
LINE 0B RESPONSE IS UNPATCHED
***LOGICAL UNIT CAPABILITY ERROR
 SBO IS NOT A WRIT DEVICE
 SLO IS NOT A WRIT DEVICE
***LOGICAL UNIT CAPABILITY ERROR INITIALLY
 SLO IS NOT A WRIT DEVICE
***SYSTEM DIRECTORY ERROR
INDEX 000F HAS INVALID REQ PRI 0004
INDEX 0014 TOO LONG FOR REQ PRI 0004
```

Option 2

```
   A       Q       I        REGISTER
FFFF     7FFFF   295F
PRI LVL WAS 000
***INTERRUPT TRAP ERROR
LINE 0A RESPONSE IS UNPATCHED
LINE 0B RESPONSE IS UNPATCHED
LINE 00 LAST INTERRUPTED 8BD1
LINE 01 LAST INTERRUPTED 2925
LINE 03 LAST INTERRUPTED B478
LINE 04 LAST INTERRUPTED 042C
LINE 05 LAST INTERRUPTED B38C
LINE 06 LAST INTERRUPTED 0C9A
LINE    0 1 2 3 4 5 6 7 8 9 A B C D E F
LEVEL   F A D B 9 A 8 6 9 9 D D 6 6 6 6
***INTERRUPT TRAP ERROR INITIALLY
LINE 0A RESPONSE IS UNPATCHED
LINE 0B RESPONSE IS UNPATCHED
LINE    0 1 2 3 4 5 6 7 8 9 A B C D E F
LEVEL   F A D B 9 A 8 6 9 9 D D 6 6 6 6
***LOGICAL UNIT CAPABILITY ERROR
 SBO IS NOT A WRIT DEVICE
 SLO IS NOT A WRIT DEVICE
***LOGICAL UNIT CAPABILITY ERROR INITIALLY
 SLO IS NOT A WRIT DEVICE
LU  04  CURRENT PARA LIST AT 291D
RC  0900
C   0000
TH  FFFF
LU  18FD
N   0000
S   0042
I/O IN PROGRESS
RETURN FOR FNR WAS 1656
RETURN FOR CNR WAS 1608
LAST ENTRY TO BE SCHEDULED
0366/    1200     2569     036A     FFF3
THERE WERE 0000 OF THE 0101 VOLATILE WORDS
   ASSIGNED
ALLOCATABLE CORE MAP
INDEX START LNGTH THRD DUMP
EMPY 1F09  0870 2D5E 0A00 60FF 487F C622 997F
0003  2845  0519 2847  08FE 605F 5807 04E2 0030
```

```
0000  1400  7FFF  0000  0001  0003  0007  000F  001F  003F  007F  00FF  01FF  03FF  07FF  0FFF  1FFF
0010  3FFF  7FFF  FFFF  FFFE  FFFC  FFF8  FFF0  FFE0  FFC0  FF80  FF00  FE00  FC00  F800  F000  F000
0020  0000  8000  0000  0001  0002  0004  0008  0010  0020  0040  0080  0100  0200  0400  0800  1000
0030  2000  4000  8000  FFFE  FFFD  FFFB  FFF7  FFEF  FFDF  FFBF  FF7F  FEFF  F0FF  FBFF  F7FF  FFFF
0040  DFFF  BFFF  7FFF  0005  0006  0009  000A  0000  0000  0000  0000  0000  0000  0000  0000  0000
0050  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0060  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0070  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0080  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
0090  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
00A0  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
00B0  0000  0000  0000  0008  0366  06AB  067A  0205  021A  0C0A  0A49  0A34  0A7A  0A94  0AC4  CA83
00C0  0000  0957  0008  0000  0121  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
00D0  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  3348  34C9  0000  0000
00E0  0000  0000  0000  0000  0001  0000  0080  0000  0000  0000  063F  014B  7DCF  2EE5  0A77  0000
00F0  0265  0004  0140  0000  0BCF  7FFF  7FFF  2EBA  0AEF  000C  0C07  0C09  0004  0004  0AD8  295F
```

Figure 11-1. Sample Dump

```
EMPY 2D5E 0009 2FAB 0000 0000 0000 0000 0000
0002  2D67 0144 2D69 08FE 6022 40FF 0822 0927
EMPY 2EAB 0010 FFFF 0000 0000 0000 0000 0000
***SYSTEM DIRECTORY ERROR
INDEX 000B HAS INVALID REQ PRI 0004
INDEX 0014 TOO LONG FOR REQ PRI 0004
SYSTEM NOT SWAPPED
JP WAS IN CORE
FILE1  FILE2  FILE3  FILE4  LOADR  BP
2D69   2847   0000   0000   0000   0000
```

Option 3

```
    A       Q       I        REGISTER
FFFF      7FFE      295F
MAX CORE WAS 7FFF WITH 2EBB TO 7FFE UNPROT
MAXSEC WAS 00003E7F
MAX CORE WAS 7FFF WITH 2EBB TO 7FFE UNPROT
MAXSEC WAS 00003E7F
PRI LVL WAS 0000
***INTERRUPT TRAP ERROR
LINE 0A RESPONSE IS UNPATCHED
LINE 0B RESPONSE IS UNPATCHED
LINE 00 LAST INTERRUPTED 8BD1
LINE 01 LAST INTERRUPTED 2925
LINE 03 LAST INTERRUPTED B478
LINE 04 LAST INTERRUPTED 042C
LINE 05 LAST INTERRUPTED B38C
LINE 06 LAST INTERRUPTED 009A
LINE      0 1 2 3 4 5 6 7 8 9 A B C D E F
LEVEL     F A D B 9 A 8 6 9 9 D D 6 6 6 6
***INTERRUPT TRAP ERROR INITIALLY
LINE 0A RESPONSE IS UNPATCHED
LINE 0B RESPONSE IS UNPATCHED
LINE      0 1 2 3 4 5 6 7 8 9 A B C D E F
LEVEL     F A D B 9 A 8 6 9 9 D D 6 6 6 6
INTRPT STACK LEVEL
-1
***LOGICAL UNIT CAPABILITY ERROR
 SBO IS NOT A WRIT DEVICE
 SLO IS NOT A WRIT DEVICE

***LOGICAL UNIT CAPABILITY ERROR INITIALLY
 SLO IS NOT A WRIT DEVICE
LU  04  CURRENT PARA LIST AT 291D
RC  0900
C   0000
TH  FFFF
LU  18FD
N   0000
S   0042
I/O IN PROGRESS
RETURN FOR FNR WAS 1656
RETURN FOR CMR WAS 1608
NUM OF SCHEDL STACK ENTRIES WAS 18
NUM OF SCHEDL CALLS STACKED WAS 00
LAST ENTRY TO BE SCHEDULED
0366/    1200     2569     036A     FFF3
THERE WERE 0000 OF THE 0101 VOLATILE WORDS
  ASSIGNED
ALLOCATABLE CORE MAP
INDEX START LENGTH THRD DUMP
EMPY 1F09  0870 2D5E 0A00 60FF 487F 0622 997F
0003  2845  0519 2847  08FE 605F 5807 04E3 0030
EMPY 2D5E 0009 2EAB 0000 0000 0000 0000 0000
0002  2D67 0144 2D69 08FE 6022 40FF 0822 0927
EMPY 2EAB 0010 FFFF 0000 0000 0000 0000 0000
***SYSTEM DIRECTORY ERROR
INDEX 000F HAS INVALID REQ PRI 0004
INDEX 0014 TOO LONG FOR REQ PRI 0004
SYSTEM NOT SWAPPED
JP WAS IN CORE
FILE1  FILE2  FILE3  FILE4  LOADR  BP
2D69   2847   0000   0000   0000   0000
FINISH SYSCOP
```

ERROR MESSAGE CORRELATION

This section gives an alphabetical listing of the error messages for the system checkout program.

```
A  Q  I REGISTER
ADDRESS IN xx WAS xxxx BUT SHOULD BE xxxx
ALLOCATABLE CORE MAP
***ALLOCATABLE CORE ERROR
***INTERRUPT TRAP ERROR
***INTERRUPT TRAP ERROR INITIALLY
***LOCORE CONSTANT ERROR
```

***LOCORE CONSTANT ERROR INITIALLY
***LOGICAL UNIT CAPABILITY ERROR
***LOGICAL UNIT CAPABILITY ERROR INITIALLY
***LOGICAL UNIT TABLE ERROR
***LOGICAL UNIT TABLE ERROR INITIALLY
***MASK TABLE ERROR
***MASK TABLE ERROR INITIALLY
***POSSIBLE LEVEL HANGUP
***SCHEDULER STACK ERROR
***SYSTEM DIRECTORY ERROR
BIT TABLE CHECKSUM ERROR
CONSIDER SWAP RATE TOO RAPID
CONSIDER UNPROTECTED I/O HANGUP
CORE USAGE CAUSED SWAP WHILE JP IN DUMP
ENTRY FOR LVL xxxx INITIALLY xxxx BUT CHANGED TO xxxx
FILE1 FILE2 FILE3 FILE4 LOADR BP
FINISH SYSCOP
FORTRAN LEVELS
FORTRAN LEVELS (ERROR)
ILLEGAL BUSY INDICATOR
IMAGE START SECTOR IS xxxx
INDEX xxxx HAS INVALID REQ PRI xxxx
INDEX xxxx TOO LONG FOR REQ PRI xxxx
INTRPT STACK LEVEL
JP LOCKED OUT FOR LIBEDT OR RECOVERY
JP NOT IN CORE
JP WAS IN CORE
LAST ENTRY TO BE SCHEDULED
LEVEL xx IS USED FOR INTERRUPTS AND IS RESERVED FOR FORTRAN
LINE 0 1 2 3 4 5 6 7 8 9 A B C D E F
LINE 0 IS NOT SETUP FOR PARITY/PROTECT
LINE xx IS SET FOR LVL xxxx BUT ABLE TO INTERRUPT xxxx
LINE xx IS SET FOR LVL xxxx BUT UNABLE TO INTERRUPT xxxx
LINE xx LAST INTERRUPTED xxxx
LINE xx LAST INTERRUPTED xxxx (INVALID)
LINE xx RESPONSE IS UNPATCHED
LU 1 NOT CORE ALLOCATOR
LU xx AND xx MATCH BUT SHARED BIT NOT SET
LU xx CURRENT PARA LIST AT xxxx
LU xx IS ALTERNATE FOR xx, BUT HAS LESS CAPABILITY
LU xx IS SHARED BUT UNMATCHED
LU xx THREAD
LU xx THREAD MAY BE BROKEN
LU xx WAS MARKED DOWN
MAX CORE WAS xxxx WITH xxxx TO xxxx UNPROT
MAX CORE WAS xxxx WITH xxxx TO xxxx UNPROT (ERROR)
MAXSEC WAS xxxxxxxx
MAXSEC WAS xxxxxxxx (ERROR)
NO VALID PHYSTB FOR LU xx
NUM OF LUS DO NOT AGREE, ASSUME xx
NUM OF SCHEDL CALLS STACKED WAS xx
NUM OF SCHEDL STACK ENTRIES WAS xx
PARTITION CORE ADDRESSES
PARTITION xx hhhh
PARTITION CORE ERROR
PARTITION IS USE
PARTITION 0 ABOVE 8000
PARTITION OUT OF ORDER
PARTITION THREADS
PENDING INPUT REQUEST FOR JP
PENDING INPUT REQUEST FOR MIPRO
PRI LVL WAS xxxx
PRI LVL WAS xxxx (ERROR)
RETURN FOR CMR WAS xxxx
RETURN FOR FNR WAS xxxx
SCHEDL STACK ENTRIES
SELECT OPTION
SIGN OFF REQUESTED OF LIBEDT OR RECOVERY
SYSCOP START
SYSTEM NOT SWAPPED
SYSTEM NOT SWAPPED BUT WAITING TO SWAP
SYSTEM USING NDISP WITH REENT FORTRAN (ERROR)
SYSTEM WAS SWAPPED
THERE WERE xxxx OF THE xxxx VOLATILE WORDS ASSIGNED
xxx IS NOT A READ DEVICE
xxx IS NOT A WRIT DEVICE
xxxx UNPROT REQ WERE ACTIVE AND STACKED AT LOC xxxx

The relocatable binary loader operates under the control of the job processor to load relocatable binary programs from peripheral devices or from the program library into available unprotected core.

The loader is read from the system library by an *LGO, *L, or * entry point name statement that sets the limits of available core to the highest and lowest addresses of unprotected memory and then places the loader into the high end of unprotected core. It can also be called by a LOADER request to the monitor.

Input to the loader consists of relocatable binary format records of variable length with a maximum of 120 characters from any peripheral device in the system. EOL statements and control statements for the job processor are also in the form of format records. These format records begin with an asterisk and terminate with a carriage return (or space if input is from a card reader); they are stored in a buffer internal to the loader in ASCII code.

Loader input is single buffered. After each input operation, the accuracy of the previous input operation is checked for input/output errors. If an error occurred, the loader terminates the operation and types a diagnostic; if no errors are detected, the loader reads the next input block. If unprotected core is not large enough to accommodate the loader, a JP05 message is printed.

Note that in this section all references to the format of the relocatable binary data (RBD) blocks are interpreted by the relocatable binary loader. For example, blocks terminating on the first zero word are true only because the loader fills its input buffer with 0s prior to each input. Users interpreting RBD blocks without the use of the relocatable binary loader must ensure that the input buffer is zero-filled prior to each input.

FEATURES OF LOADER

The following are features of the relocatable binary loader:

- The object code may be patched with 15- or 16-bit addresses, depending on the program's execution base address.
- The command sequence and loader tables are paged to mass storage if available core becomes filled. This alleviates the problem of loader table overflow while loading large programs with many entry points. Loading can take place if 195 words plus the length of the loader are available.
- Both the core-resident entry point tables (refer to Unprotected Common, section 2) and the program library may be linked to in any order without a duplicate entry point error.
- Two programs may reference the same external name in different addressing modes; one with relative addressing, the other with absolute addressing.
- Control and update information contained in the NAM block is printed.

PARTITION LOADING

TRANSFER ADDRESS CONSIDERATIONS

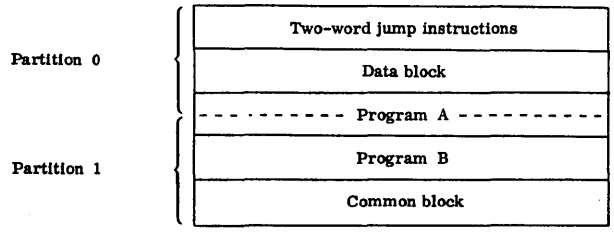
A two-word absolute jump instruction is always stored as the first two words of a command sequence during a partition load. The last transfer address encountered is stored as the second word of the jump instruction. This satisfies the requirement that the first word of a system directory program must be executable. The absence of a transfer address is considered an irrecoverable error by the loader.

DATA AND COMMON DECLARATIONS

Data (labeled common) and common blocks may reside within the partitions used by the programs loaded under an *A statement to LIBEDT. The loader places the data block in front of the first program declaring data. The common block resides at the end of the last partition used.

Example:

Programs A and B are loaded by an *A under LIBEDT and occupy two partitions. Program A is declared both a data block and a common block.



Programs loaded in partitions may also link to the system data and system common blocks.

RELOCATABLE BINARY INPUT

Blocks of ASCII are identified by an asterisk in bits 15 through 8 of the first word. If bits 15 through 8 are anything else, relocatable binary is assumed.

The driver for the input device verifies that the block is read correctly.

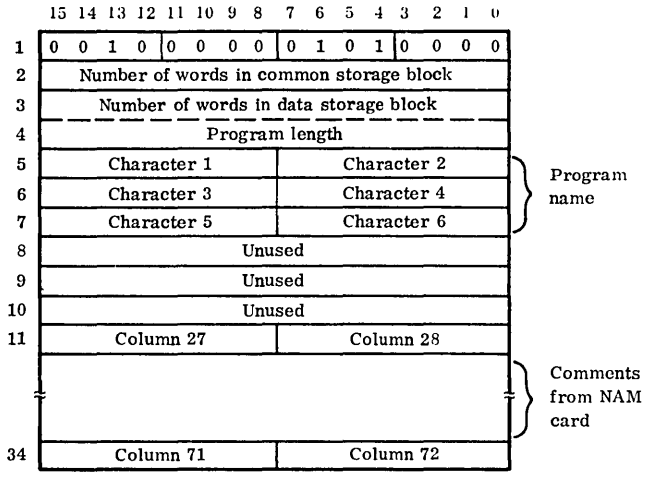
The loader recognizes relocatable binary blocks by the type of indicator field in bits 15 through 13 of the first word of the block. If the loader is unable to recognize the indicator, it does not process the block. The following block types are defined:

Type	Indicator	Description
NAM	001	Name block
RBD	010	Command sequence block
BZS	011	Zero storage block
ENT	100	Entry point block
EXT	101	External name block
XFR	110	Transfer address block

Input for a single relocatable binary program must begin with a NAM block and terminate with an XFR block. There must be only one NAM block and one XFR block. The EXT blocks must follow the RBD blocks; RBD, BZS, and ENT blocks may be in any order. If input consists of several relocatable binary programs, the NAM block of the third program must follow the XFR of the second, etc.

NAM Block

The NAM block contains a word count for common storage and data storage, the program length, and the name of the program.



NAM is the first block in each relocatable binary program loaded. An out-of-order NAM block is catastrophic and results in a diagnostic and a loader exit.

Words 11 through 33 contain name information. These words are printed by the loader during load operations and initialization and contain comments preset by the user on the NAM card.

The loader uses the NAM block to allocate available core to common and data storage. When several relocatable binary programs are to be loaded in one operation, the loader assigns space according to the first NAM block processed that specifies common storage. Data storage is assigned in the same manner. Subsequent NAM blocks must not require larger common or data storage than the first common or data storage to be declared. The loader recognizes violation of this rule as an error. As each NAM block is read by the loader, the core location, which is the base address of the program's command sequence, is printed on the list output device together with the program name.

Execution time core is allocated for relocatable binary input as follows:

Common storage†	Assigned at the high end of execution time memory
Command sequence storage	Assigned at the low end of execution time memory; relocatable binary programs are loaded in a forward direction. All assigned execution time is available for command sequence storage. If assigned command sequence limit is exceeded, the loader terminates the operation and issues an error message.
Data storage†	Assigned space in core is reserved for command sequence storage; data storage precedes command sequence storage for the first program declaring data storage on its NAM block.

RBD Block

An RBD block contains a portion of the actual command sequence data of the program.

Words 2 through 59 contain the relocation bytes and words for the command sequence input.

Each relocation byte is a 4-bit indicator that identifies a word of the command sequence input as an absolute 15-bit address or as a 15-bit address relative to some relocation base. The relocation base for a word is determined by the particular combination of bit settings within the relocation byte.

The following are the relocation bytes in the RBD blocks:

0000	Absolute (no relocation)
0001	Positive program relocation
0101	Negative program relocation†
0010	Positive common storage relocation

†Common and data may also be assigned to system common and system data addresses.

0110	Negative common storage relocation†
0011	Positive data storage relocation
0111	Negative data storage relocation†

The following is the core image of the RBD block:

	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
1	0 1 0 0	0 0 0 0	0 1 0 1	0 0 0 0
2	R0	R1	R2	R3
3	W0			
4	W1			
5	W2			
6	W3			
7	R4	R5	R6	R7
8	W4			
9	W5			
10	W6			
11	W7			
12	R8	R9	R10	R11
≈				
52	R40	R41	R42	R43
53	W40			
54	W41			
55	W42			
56	W43			
57	R44	R45	Not used	
58	W44			
59	W45			
60	Not used			

Where: W0 is the origin address of the input block.

R0 is the relocation byte for W0.

Wn is the nth word of the input block (n = 1 through 45)

Rn is the relocation byte of the nth word.

There is one relocation byte for every word in the command sequence input and a maximum of 45 entries in the RBD block. The first word is the address relative to the start of the program where the loader begins storing command sequence data. The relocation byte for the first word address (storage address) of an RBD block may be 0000, 0001, or 0011. If the field contains a number larger than 0011, 0011 is assumed. Zero is the loading bit for all but the last relocation byte; one is the leading bit for the last relocation byte. The loader accepts absolute origins within the area from $C5_{16}$ to $E3_{16}$ only.

In processing an RBD block, the loader picks up the 16 bits that represent the first word address of the command sequence data in the block. It adjusts this address for relocation, according to the setting of the bits representing its relocation byte. The resulting absolute address is the first word address in core to receive the command sequence data (stored in consecutive locations) at the completion of loading. Each word is relocated according to its relocation byte.

In the following rules for address relocation, the letter E identifies the core address after adjustment for relocation.

E = A	Absolute addressing
E = P + M	Positive program relocation
E = -(P + M)	Negative program relocation††
E = C + K	Positive common storage relocation
E = -(C + K)	Negative common storage relocation†
E = D + N	Positive data storage relocation
E = -(D + N)	Negative data storage relocation††

Where: A is the absolute address.

P is the positive program relocatable address.

C is the positive common storage relocatable address.

D is the positive data storage relocatable address.

M is the base address for the program's command sequence storage.

K is the base address for common storage.

N is the base address for data storage.

A negative relocation base is represented by the ones complement of the positive relocation base.

† Illegal if execution time address is in part 1.

†† Negative relocation is not allowed in part 1 programs.

The first word address for the command sequence storage of an RBD block is positive program relocatable, absolute, or positive data storage relocatable. The only allowable absolute first word address for command sequence storage on an RBD block is within the range of $C5_{16}$ to $E3_{16}$ (the FORTRAN scratch pad area). All other absolute addresses generated with an ORG statement are rejected by the loader, an error message issued, and the load terminated. The loader computes the absolute value for the execution time storage address according to the rules above.

Command sequence data is absolutized sequentially for execution time addresses beginning with the first word address specified in the RBD block. Before this data is absolutized, the loader checks that the absolute first word address is greater than the lower limit of available core and less than the upper limit. If not, an error message is issued and loading is terminated.

Execution time core is defined as the areas of core that are available for the load. For an unprotected program, this is the area designated by the contents of $F6_{16}$ and $F7_{16}$ low core locations; for partitions, it is the beginning of the first partition and the end of the last partition designated in the LIBEDT control statement.

BZS Block

A BZS block contains relocation bytes, starting addresses, and block sizes for areas of core to be cleared to zeros when the program is loaded.

The relocation byte for a starting address may be 0000, 0001, or 0011. These relocation bytes have the same meaning as described for the RBD block (refer to the RBD Block section). In a BZS block entry, the loader adjusts A for the relocation base and stores zeros at every address in memory from A to A + S - 1. The relocation bytes for all but the last entry in the BZS block have a zero leading bit. The relocation byte for the last entry has a one leading bit. A maximum of 25 entries are contained in the block beginning with word two and extended to word 58; words 59 and 60 are not used.

When processing a BZS block, the loader picks up a starting address, a relocation byte, and a block size for a number of sequential locations to be cleared to zero at load time. If the starting address is not absolute, the loader adds the relocation base and stores zeros at sequential locations determined by the block size and beginning with the starting address. The rules for address relocation are the same as for an RBD block and the same checking procedures are used.

The following is the core image of the BZS block:

	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
1	0 1 1 0	0 0 0 0	0 1 0 1	0 0 0 0
2	R1	R2	R3	R4
3	A1			
4	S1			
5	A2			
6	S2			
7	A3			
8	S3			
9	A4			
10	S4			
11	R5	R6	R7	R8
≠				≠
47	R21	R22	R23	R24
48	A21			
49	S21			
50	A22			
51	S22			
52	A23			
53	S23			
54	A24			
55	S24			
56	R25	Not used		
57	A25			
58	S25			
59	Not used			
60	Not used			

Where: An is the starting address of the nth entry.

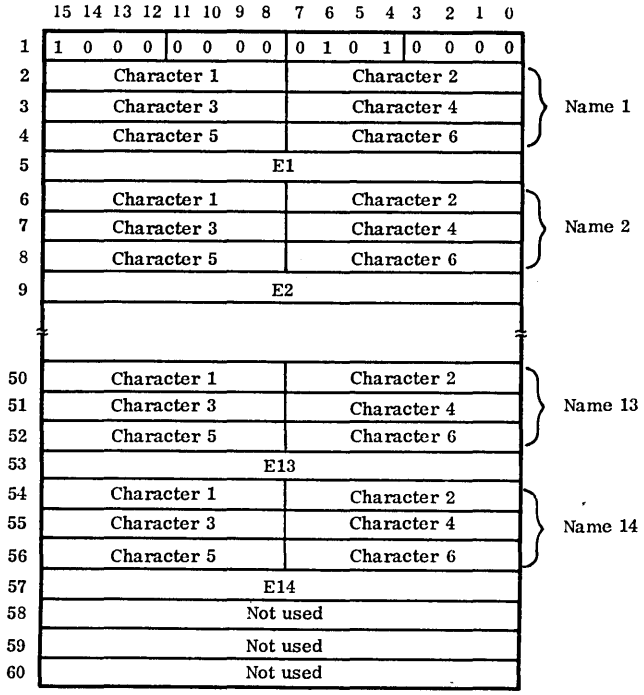
Sn is the size of the BZS reservation for the nth entry.

Rn is the relocation byte of the nth entry.

ENT Block

Up to 14 entry point names and addresses may be included in an ENT block. The end of data in this block is identified by zeros. If the sign bit of a word containing the entry point address is zero, the address is program-relocatable. If the sign bit of the word is one, the address is absolute and in ones complement. Data begins in word 2 and extends to word 57; words 58, 59, and 60 are not used.

The following is the core image of the ENT block:



Where: Name n is the six-character name of the nth entry block.

En is the entry point address of the nth name. En is negative (ones complement) if absolute and positive if program-relocatable.

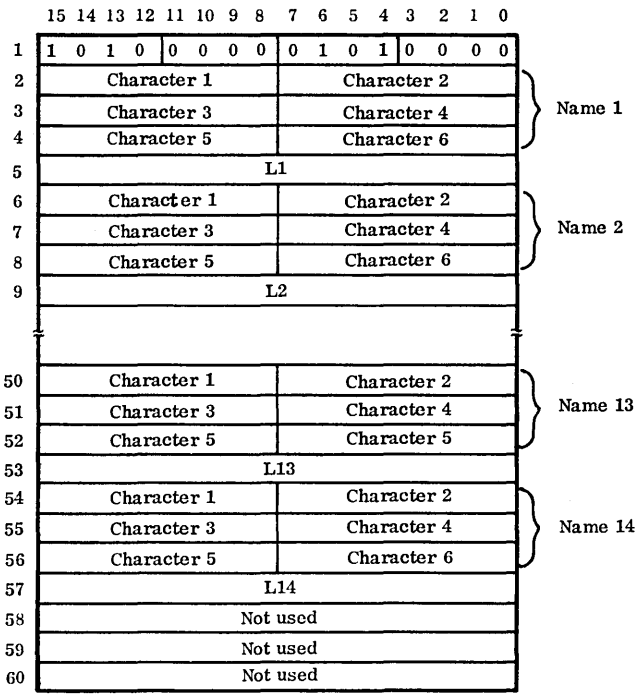
When processing an ENT block, the loader records the entry point name in its table. The entry point address is adjusted for relocation (either program or absolute) and then recorded in the table of entry points. This procedure is repeated until the end of input is reached (a name equal to zero).

For each name, the loader determines if an entry point has been previously recorded in the table. If so, a duplicate entry error has occurred. The same entry point name may not be used by two programs to occupy memory space at the same time. Only the first occurrence of an entry point name is valid; others are illegal and are not loaded.

EXT Block

Up to 14 external names and link addresses may be included in an EXT block.

The following is the core image of the EXT block:



Where: Name n is the six-character name of the nth entry block.

Ln is the link address of the nth name. Ln is negative (ones complement) if absolute and positive if the program is relocatable.

Zeros indicate the end of the EXT block. If the sign bit of the word containing the link address is zero, the address is program-relocatable. If the sign bit is one, the address is absolute and in ones complement. The format of the data in the block is the same for EXT as for ENT information.

The loader records the external name in its table. The link address for the external name is adjusted for relocation (program or absolute) and recorded in the table of external names. This procedure is repeated until the end of the input block is reached (a zero is encountered).

XFR Block

The XFR block contains a transfer address (in words 2 through 4), which is six ASCII characters in length including

trailing spaces. The transfer address must be an entry point in the program being loaded or in another program loaded during the same load operation.

The following is the core image of the XFR block:

	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
1	1 1 0 0	0 0 0 0	0 1 0 1	0 0 0 0
2	Character 1		Character 2	
3	Character 3		Character 4	
4	Character 5		Character 6	

The XFR block must be the last in a relocatable binary program. If an XFR block is out of order, a loader error message is issued and the load is terminated. The loader records the transfer address in the XFR block. If two or more relocatable binary programs are loaded with one operation, the loader saves the last transfer address for the start of execution.

NONRELOCATABLE BINARY INPUT

Input to the loader may not always be relocatable binary blocks. It may be in ASCII format (e.g., EOL, system control statements).

If bits 15 through 8 of the first word in the block are set to the ASCII code for an asterisk, information is stored in the input buffer in ASCII code. The end of a nonrelocatable binary input record is the internal code for a blank or a carriage return.

EOL BLOCK

The EOL block, which marks the end of loader input, contains an asterisk followed by a T in the first word, as shown below. The normal procedure for termination implies that the operation has been error-free.

The following is the core image of the EOL block:

* = 00101010	T = 01010100
CR = 00001101	Not used
Not used	

CONTROL BLOCK

Control blocks are similar to the EOL block and are stored in the loader's buffer in ASCII code. They are not fixed in length and are terminated by a blank or carriage return. These blocks are handled by the job processor rather than the loader. The loader transfers control to the job processor, giving it the address of the input buffer in A. *L and *X are examples of operating system control information blocks.

*PAGE STATEMENT

When the loader encounters a *PAGE statement, the load address is incremented to the start of the next 2K (2048 words) page in core. This statement is for use only by systems that use the paging feature of the CYBER 18 (for instance, ITOS).

The library editing program allows the user to:

- Add a program or file to the program library
- Remove a program or file from the program library
- Replace one program or file with another in the program library
- Replace allocatable core programs or partition core programs in the system library
- Combine several relocatable binary programs in an absolute binary record and output this record on the binary output device
- Transfer information between peripheral devices and/or job processor file manager files
- Set request priorities for system directory programs

The system library is composed of programs stored in absolute form. Each program in the library is referenced by an entry in the system library directory. The program identification is the directory ordinal that identifies the first location containing information relating to the program (refer to appendix D).

The program library is composed of programs stored in either relocatable or absolute form. Each relocatable binary program in the library is referenced by one or more entries in the program library directory. These entries consist of all entry points declared in the relocatable binary program. No two programs in the program library may have duplicate entry point names, although a file may have the same name as an entry point in a program. Each file is referenced by a file name in the program library.

The system library directory cannot be expanded since it resides in core. However, empty spaces can be provided during initialization by making an *Y or *YM entry for programs which do not exist. An *Y statement, when added as a dummy, must have a dummy program to save core to load into when updated by LIBEDT. The library editing program cannot predict interaction of changed programs, especially while the change is taking place; this responsibility remains with the user.

LIBEDT PROGRAM

The library editing program is stored in the system library by the System Initializer. The control statement *LIBEDT instructs the job processor to load the library editing program into protected core and begin operation. The library editing program types LIB on the comment device to indicate it has been entered and, after completing its functions, IN is typed on the comment device. LIBEDT uses the system loader (see section 12) for any program load operations.

The library editing program outputs to three devices:

- Comment device – Prints error messages and indicates entrance to the library editing program
- Standard binary output device – Produces absolute records on an external device from relocatable binary input
- Standard print output device – Lists the system or program library directory

CONTROL STATEMENTS

Control statements to the library editing program are format records. The first character of a control statement must be an asterisk; the last must be a carriage return. Intervening characters identify the type of statement and action.

The library editing program contains a list of acceptable control statements. Control statements read by the program must match an entry in this list. Each entry in the list is associated with an address to which the library editing program is transferred when carrying out the operation directed by the control statement.

Table 13-1 is the standard list of control statements for the library editing program.

LIBEDT, with all the preceding functions, resides in the system library as one program. As each particular function of LIBEDT is called, that function processor along with associated subroutines are read into core under LIBEDT control for execution.

Additional control statements may be processed by adding the control statement and by supplying a sequence of code in the program to accomplish the required action in a manner similar to that for the existing LIBEDT functions.

The *F and *FOK statements are pseudo statements; therefore, no processor or subroutines exist for these statements.

*M — REPLACE PROGRAM

An *M statement replaces a program in the system library that executes in allocatable core with another program. Programs that declare blank and/or labeled common link to the system common.

TABLE 13-1. CONTROL STATEMENTS FOR LIBRARY EDITING PROGRAM

Input Statement	Transfer Address to Begin Action	Input Statement	Transfer Address to Begin Action
*M	SLINSN	*N	FILE
*L	PLINSN	*S	RPUPDT
*P	PGHABS	*T	COPY
*U	CONCTL	*K	CHANGE
*V	SYSINP	*R	REMOVE
*Z	LIBXIT	*A	AINSN
*DM	LISTSD	*F	
*DL	LISTPD	*FOK	

The following is the statement format:

*M,or,s,d,M,N

Where: or is the ordinal number in the system library directory; it is a required parameter. If the ordinal number does not appear in the system library directory, or if the parameter is not specified, the statement is illegal.

s is a mass storage address; this parameter is illegal if M is blank. After the relocatable binary programs are loaded and linked, the thread of the directory entry being replaced is checked. If it is busy, LIBEDT waits for it to be freed, indicating that this directory entry is not currently being operated. The thread is then set busy to prevent the file from being scheduled while LIBEDT is manipulating the directory and writing the new file onto mass storage, beginning at address s. If s is not specified in the input statement, mass storage is searched for the first block of sectors large enough to contain this file. In most instances, this is at the end of the library. When the new program is shorter than the one it is to replace, it is stored on the sectors of the file it is replacing. At the completion of the update, the thread is cleared.

d is not used and must be blank.

M is the mass storage indicator. When M appears in the statement, the program to be replaced is mass-storage-resident and the ordinal represents its position in the system library directory relative to other mass-storage-resident programs. When the length of the replacement program is less than or equal to the length of the program being replaced (rounded to the nearest sector), the new one overlays the old one on mass storage.

If the replacement program is longer than the program to be replaced, it is added to the library on the first block of available sectors large enough to contain the file. The total length of the file being loaded cannot exceed the length of allocatable core, including unprotected core.

If M is omitted in the statement, the program to be replaced is core-resident and the ordinal represents its position in the system library directory relative to other core-resident programs. The length of the new program, if core-resident, must be equal to or less than that of the program being replaced. Since the directory entry for a core-resident ordinal does not contain length, no error indication can be given if it is longer than the program being replaced.

CAUTION

On-line use of LIBEDT for replacement of system library programs exposes the operation to the following potential faults:

- The interrupt system is disabled during core-resident program replacement. Thus, for large programs, the system response could be inhibited for excessive periods of time.
- The program length of a core-resident program being replaced cannot be checked by LIBEDT. A larger program is loaded without error, potentially destroying part of the system.

N indicates that linking to the program library is not required. When a new program is added to the system library, the library editing program issues a loader request to load one or more relocatable binary programs from the standard

input device until a loader EOL statement (*T), a nonloader statement, or a device failure is detected. If any unpatched externals exist at this time, automatic linkage is performed first to the CREP table and then to the CREP1 table. Any unpatched externals that remain after this linkage are listed and the user has the option of continuing by typing an *. Termination of the load is performed by entering an *T.

If the field is blank, automatic linkage to the program library is performed after the CREP linkage. Any remaining unpatched externals are listed and the user may continue or terminate as described above.

CAUTION

Run-anywhere FORTRAN programs cannot specify part 1 core-resident, upper-bank entry points as formal parameters in subroutine calls.

If loading is terminated with an EOL statement, the library editing program looks to the comment device (*U statement) or the standard input device (*V statement) for the next control statement. If loading was terminated by a nonloader statement, the nonloader statement is processed as a control statement to the library editing program.

*L — ADD/REPLACE PROGRAM

An *L statement adds a new program to the library or replaces a program in the library.

The following is the statement format:

*L,epn

Where: epn adds a new program to the library if the entry point name does not appear in the program library directory.

When an addition is made to the library, the library editing program reads format records of binary input from the standard input device and writes them onto mass storage.

Entry point names for programs added to the library are recorded in the directory together with the beginning mass storage addresses.

If the entry point name does appear in the directory, the program containing this entry point is replaced. The new entry point name is placed in the directory. When a program is replaced in the library, its entry point name is removed from the directory.

If a mass storage unit is to be used as a system input device, the input operation begins at the first scratch sector. This feature allows the user to assemble and obtain load-and-go output. By assigning the load-and-go unit as the system input device with a monitor control statement (*K,I unit number), the load-and-go unit becomes the input device for processing an *L, entry point name control statement.

*P — PRODUCE ABSOLUTE RECORD

An *P statement directs the library editing program to produce an absolute record from one or more relocatable binary programs. The relocatable binary programs are loaded in core by the loader under control of the library editing program.

The following is the statement format:

*P,n,R/P,sa

Where: n is the record format parameter.

n ≠ F or omitted	A single format record is written on the standard binary output device.

CAUTION

Some binary output devices (for example, magnetic tape drives) are not able to write single format records longer than a value specified in the driver. The driver may truncate a longer record or segment and write it as a series of physical records. The user should refer to the appropriate MSOS 5 peripheral equipment reference manual.

n ≠ F	Output is in format records of 96 words each. If binary output is assigned to a mass storage device, this unit must be the library unit, as subsequent operations except the absolute file to start on the scratch area following the library (that is, the sector defined by the contents of $C0_{16}$ and $C1_{16}$).

R/P is the order of linkage parameter. The use of the R/P option makes it possible to build subprogram parts for allocatable foreground or to partition core programs. Such programs can then be stored on the disk as files by using the *N LIBEDT processor, which is overlaid in the foreground user buffer areas by the user programs. The use of this technique negates the need for using system directory entries for such files.

- P parameter – If the P parameter field is blank, the order of linkage is the preset table, the program library, and/or the unprotected unlabeled common area. If any unpatched externals exist at this time, the *P processor links to the CREP table first and then to the CREP1 table. If any unpatched externals still exist following this linkage, they are listed and the user can enter an * to continue or *T to terminate.

  If the parameter field is set to P, linkage is performed in the following order: protected unlabeled COMMON, CREP table, CREP1 table, and the program library. If unpatched externals exist following the linkage, a list is printed and the user can enter an * or an *T.

- R parameter – If the parameter field is set to R, linkage is performed in the following order: protected unlabeled COMMON, CREP1 table, CREP table, and the program library. If unpatched externals exist following the linkage, a list is printed and the user can enter an * or an *T.

NOTE

n all preceding cases of the R/P parameter, absolutizing begins at the location specified by $F7_{16}$ plus one.

- If the parameter field is set to a numeric value between 1 and 16, the relocatable binary programs are absolutized at the beginning of the partition specified by the field.

In all other cases of the R/P parameter, absolutizing begins at the location specified by $F7_{16}$ plus one.

sa is the starting address and can be one of the following:

$hhhh_{16}$	Hexadecimal number as core address
Entry point name	Core address for an entry point of a relocatable binary program read in by the loader. The entry point name DATBAS is used to reference the data block set aside during a loader operation.
Entry point name + hhhh	hhhh is added to or subtracted from the core address to find the starting address.

If the starting address is specified, the binary output extends from the starting address to the last word of the load. Therefore, the starting address must not be specified beyond the last word address of the relocatable binary load.

If the *P statement is unacceptable to LIBEDT because it exceeds the last word address of the relocatable binary load, the error message E11 appears on the print device. The operator must type in an acceptable starting address without repeating an *P,n. A carriage return without a starting address has the same effect as a starting address equal to the contents of location $F7_{16}$ plus one. The error message is issued by the loader processing the starting address portion of an *P statement.

*U — GET NEXT CONTROL STATEMENT

An *U statement directs the library editing program to go to the comment device for subsequent control statements.

The statement has the following format:

*U

*V — GET NEXT CONTROL STATEMENT

An *V statement directs LIBEDT to read control statements from the specified logical unit until an *U statement is read.

The statement has the following format:

*V,lu,m

Where: lu is the logical unit; if the logical unit is not specified, LIBEDT (system) input unit is assumed.

m is the mode of the control statement

A Formatted ASCII mode
B Formatted binary mode

If mode is not specified, the control statements are read in formatted ASCII mode.

If the control statement *LIBEDT is read under an *V option in the job processor, LIBEDT continues processing with the same *V option.

*Z — TERMINATE PROCESSING

An *Z statement terminates the library editing program processing and returns control to the job processor.

The statement has the following format:

*Z

*DM — LIST SYSTEM LIBRARY DIRECTORY

An *DM statement directs the library editing program to list the system library directory on the print output device and includes an *Y or an *YM ordinal at the beginning of each line.

The statement has the following format:

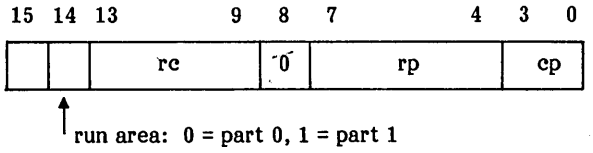
*DM

The format of the dump is n lines of hexadecimal numbers as shown:

Group	0	1	2	3	4	5	6	7
n lines	n	xxxx	xxxx	xxxx	xxxx	xxxx	0000	xxxx
	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.
	n	xxxx	xxxx	xxxx	xxxx	xxxx	0000	xxxx

Where: n is the ordinal of the system library program ($n \leq 256$). One line is provided for each *YM statement:

word 1 has the format:



Where: rc = Request code
rp = Request priority
cp = Completion priority

word 2 = Lowest possible execution address in core

word 3 = Thread address

word 4 = Q register contents when program is entered

word 5 = Program length in words

word 6 = 0

word 7 = Mass storage address of program

The form of the directory entry is shown in appendix D.

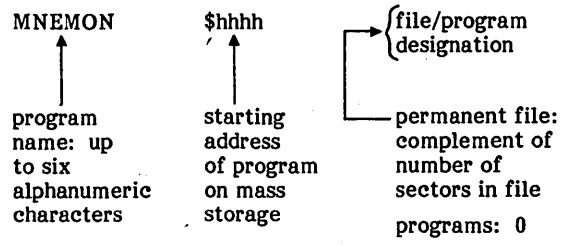
*DL — LIST SYSTEM LIBRARY DIRECTORY

An *DL statement directs the library editing program to list the program library directory on the standard print output device.

The statement has the following format:

*DL

The format of the dump is n lines with each line as shown:



Where: one line is supplied for each program in the directory.

*N — MODIFY PROGRAM LIBRARY

An *N statement is used to add, replace, or edit a permanent binary file in the program library. When a file is added or replaced, only the name and mode need to be specified.

The statement has the following format:

*N,n,w_1,w_2,m

Where: n is the name of the file; the name is a one- to six-character identification by which the file is addressed.

w_1 is the first word of the file to be changed. If only part of a file is to be changed, w_1 and w_2 (where w_1 is less than w_2) are used.

w_2 is the last word of the file to be changed (refer to w_1). If w_2 is omitted, only the word specified by w_1 is changed.

m is the mode of input.

- A ASCII format records
- B Binary format records

Input to an *N processor consists of format records of 96 words or less.

Input is terminated with any valid LIBEDT or *Z statement.

The load-and-go unit can be used as an input device for processing an *N statement in the same way as an *L statement (see the *L section).

*S — SET CORE REQUEST PRIORITY

An *S statement sets the core request priority of an entry in the system directory. This is the priority of the area of core where the file runs.

The statement has the following format:

*S,or,v,M

Where: or is the ordinal number that refers to an entry in the system directory.

v is the level at which the request priority is to be set. $0 \leq v \leq 15$.

M indicates that the ordinal number is mass-storage-resident. If not set to M, the ordinal number is core-resident.

*T — TRANSFER INFORMATION

An *T statement permits the transfer of information between any two peripheral devices such as card-to-tape or tape-to-printer. An *T can also be used to transfer information to, from, or between job processor file manager files.

An *T statement can be used in conjunction with an *F pseudo LIBEDT statement to perform information transfers within a batch job. Upon recognizing an *F during a transfer operation, LIBEDT outputs the IN message and proceeds to read the next control card.

If the file manager is present in the system and job processor files are used, LIBEDT can be used to perform library functions with those files, provided they have been assigned by an *OPEN statement in the job processor.

Example:

Call	*JOB	Call job processor.
	*OPEN,FILEA,1234,W,20	Assign files as input.
	*LIBEDT	Call LIBEDT.
	*V,20	Go to FILEA for input statements.
Typical Uses	*T,20,B,7,B	Transfer information from FILEA to logical unit 7.
	or	
	*L,PROG	Put relocatable program previously punched on FILEA on library.
	or	
	*N,PROG1, , , B	Put absolutized file punched on FILEA on library.
	or	
	*M,10, , , M	Put allocatable core program from FILEA on system library.
	or	
	*A,12,2,1, , , , ,	Put partition core program from FILEA on system library

The following is the format for an *T statement:

*T,i,mi,o,mo,n,f

Where: i is the input logical unit; if omitted, LIBEDT's standard input binary unit is selected.

mi is the mode of input.

A	ASCII
B	Binary

o is the output logical unit; if omitted, LIBEDT's standard output unit is selected.

mo is the mode of output.

A	ASCII
B	Binary

n is the upper limit on the number of records to be transferred. If n is omitted, records are transferred until the input device is empty or fails, encounters an *F control statement, or the number of files specified is reached. The upper limit of the number of records to be transferred is decimal. At the end of transfer the number of records and files encountered is printed in decimal format and output on the standard print device.

f is the upper limit on the number of files to be transferred. If f is omitted, files are transferred until the input device is empty, fails, encounters an *F control statement, or the number of records specified is reached. The upper limit of the number of files to be transferred is decimal. At the end of transfer the number of records and files encountered is printed in decimal format and output on the standard print device.

*K — CHANGE DEVICES

An *K statement, which may occur in any order with respect to other statements, allows the operator to change LIBEDT devices. These changes are internal to LIBEDT and do not affect the system device assignments.

The parameters of an *K statement may be in any order, but must be separated by commas. The statement *K,I2,L5,P3, sets the LIBEDT input unit to logical unit 2, its print unit to logical unit 5, and its binary output unit to logical unit 3.

An *K statement is terminated by a carriage return.

The following is the format for an *K statement:

*K,Ilu,Plu,Llu

Where: lu is the logical unit number. An error exit is taken if a unit number designates a protected device or a print unit specifies a mass storage device.

I is the LIBEDT input unit.

P is the LIBEDT binary output unit.

L is the LIBEDT print unit.

*R — REMOVE PROGRAM

An *R control statement removes a program with entry point n from the program library.

The statement has the following format:

*R,n,F

Where: n is the entry point of the program name of the file to be removed. If F is included, the file name n is removed.

F specifies that n is a file name.

*A — REPLACE PARTITION PROGRAM

An *A statement replaces a partition program in the system library with another partition program.

The program to be replaced is mass-storage-resident. The ordinal represents its position in the system library directory relative to other mass storage programs. When the length of the replacement program is less than or equal to the length of the program being replaced (rounded to the nearest sector), the new program overlays the old one on mass storage. If the replacement program is longer than the program to be replaced, mass storage is obtained from the first area of the disk large enough for the file. In most cases this is at the end of the library.

Next, the loader request is issued to load one or more relocatable programs from the standard input device until a loader EOL statement, a nonloader statement, or a device failure is detected. If the L parameter was specified, the program interrogates the user by typing * on the output comment medium. If the user wants to load from another input logical unit, the reply is *K,Ilu, and a carriage return. The program continues until is is terminated again. If the user is through loading when an * interrogation message is sent requesting another logical unit, the user must type an *T and press RETURN. The loading process terminates.

If any unpatched externals exist at this time, the program links:

- Other programs just loaded
- The CREP1 table, if present
- The CREP table
- The program library if the P parameter is set

Any unpatched externals at the end of this load are listed; to continue, the user must type an * and press RETURN.

After the binary programs are loaded and linked, a check is made on the thread of the directory entry being replaced. If busy, LIBEDT waits for it to be freed, indicating that this directory entry is not currently being operated. The thread is then set busy to prevent scheduling of the program while LIBEDT is manipulating the directory and writing the next program on mass storage. At completion of update, the thread is cleared.

If loading is terminated with an EOL statement, the library editing program looks to the comment device (an *U statement) or standard input device (an *V statement) for the next control statement. If loading was terminated by a nonloader statement, the nonloader statement is processed as a control statement to the library editing program.

The system library directory cannot be expanded since it resides in core. However, empty spaces can be provided during initialization by making an *YM entry for programs that do not exist. The library editing program cannot predict interaction of changed programs, especially while the change is taking place. This responsibility remains with the user.

The following is the format for an *A statement:

*A,ord,s,n,d,c,l,p,m

Where: ord is an ordinal of the program being replaced in the system library directory. If the ordinal number is not in the system library directory, or it is not specified, the statement is illegal.

s is the starting partition number, 0 through 15; if omitted, the statement is illegal. Partition 16 may be indicated if unprotected core is in part 1. If partition 16 is used, the n parameter must be one or not used. Partition 16 is unprotected core and programs are absolutized starting at $F7_{16}$ plus one. The user must schedule any program absolutized for execution in partition 16 at a completion priority greater than two.

n is the number of partitions the program requires to load a program. The range of n is 1 through 16; if n is omitted, one partition is assumed.

d is the data base indicator; this is an alphabetic indicator, and if omitted, data area within the program is created. If it is equal to d, the system data base is used.

c is the common indicator; this is an alphabetic indicator, and if omitted, common area within the partition of the program is created. If specified, the common area in system common is used.

l is the multiple input units indicator; this is an alphabetic indicator, and if omitted, loading is performed from only one logical unit. If specified, the system interrogates the user for new input logical units. This interrogation is repeated when the new input unit has completed loading and continues until the user terminates the load.

p is the link program library indicator; this is an alphabetic indicator, and if omitted, the program library is not linked to loaded programs.

m is the memory map indicator; this is an alphabetic indicator, and if omitted, no memory map is printed.

*F — END-OF-TRANSFER INDICATOR

The *F statement is a pseudo instruction to the *T processor of LIBEDT. When an *F followed by two spaces is encountered, the current *T operation is terminated and the next LIBEDT control statement is read from the standard input device.

*FOK — TRANSFER INDICATOR

This statement, like the *F, is a pseudo instruction to the *T processor of LIBEDT. If this statement is encountered at input, an *F is transferred to the output device. This allows for use of *F on devices such as magnetic tape, paper tape, etc.

To increase the ease of installing, updating, and debugging programs, the CYBER 18/1700 MSOS has system and program maintenance routines. These routines are discussed in this chapter.

CALLING STATEMENTS

The following list gives the calling name for the various routines which are described in sequence in this section.

INITIALIZER	
SILP	System initializer loading program
LIBRARY PREPARATION	
SKED	Skeleton editor
LIBILD	Library builder
LIBMAC	Macro library preparation routine
SETPV4	CYBER 18/1700 editing tape and update program (SETUP)
LIST AND SORT	
LULIST	Logical unit listing
LISTR	List relocatable
EESORT	Entry point/external sort and list
OPSORT	Operand sort
PROGRAM COMPRESSION	
CYFT	COSY format
COSY	Source program compression
LCOSY	List COSY
I/O UTILITIES	
DTLP	Disk-to-tape loading program
EDTLP	Extended DTLP
DSKTAP	Disk-to-tape program
IOUP	Input/output utility package
FILE EDITING	
EDITOR	Word processing text editor
PROGRAM TRACE	
TRACE	On-line trace

All of these programs can be called from the job processor using at *NAME statement.

SYSTEM INITIALIZER LOADING PROGRAM (SILP)

The system initializer loading program provides a means of loading the system initializer into core. The system initializer must be absolutized and put on the disk prior to using SILP with a file name SI.

The procedure to execute this program is:

```
*JOB
J
*SILP
```

If unprotected core is in the upper 32K, SILP types:

```
THE SYSTEM INITIALIZER WILL BE MOVED TO
LOCATION 4000 AND EXECUTED

TURN OFF PROTECT SWITCH AND TYPE CARRIAGE
RETURN
```

If unprotected core is in the lower 32K, the system initializer is executed where it is brought into core.

NOTE

The System Initializer must be loaded into the lower 32K of core.

SETUP (SETPV4)

The CYBER 18/1700 editing tape and update program (SETUP) provides a user with the capability of building and maintaining installation materials for all CYBER 18/1700 products. It increases the ease of handling PSR updates and binaries of user modifications that are made to the system and other associated software.

SETUP allows the user to make changes to installation materials in any form (paper tape, magnetic tape, or cards). It is a mechanism to add, remove, or change the binaries comprising an installation tape or deck. It has the advantage of being able to handle products, such as MSOS FORTRAN, which have the same program name but unique programs contents. A control card deck or tape that is used to build a system can be saved and reused when reconstruction becomes necessary.

THEORY OF OPERATION

The SETUP program must be able to determine the type of update it is doing and where to find the working materials, which is determined by an *L control statement. An *C control statement permits the user to know the position of each type of entry (binary control statement, relocatable binary data, absolute binary information, and ASCII records) relative to other units. SETUP works on a positional theory and internally identifies an entry with respect to its position from the beginning of the information (tape, deck, etc.).

As the program is trying to insert, delete, or replace, it expects a new binary to be next on the binary unit, unless informed to an * that the last binary is to be used again.

Control statements must be in the order of their respective references on the master unit. If these statements are not in the correct order, the control statements are sorted into the correct order. Updates on the binary unit must be in the order of their respective references on the master unit; the updates are not sorted. When the control set contains *S commands, the control statements are not sorted. They are executed in the order input.

All control information is given with respect to the master unit. All *I, *D, and *R statements alter the information on the master unit; during use of an *S statement the master unit is still used for reference. Although it appears that selection is made freely from both binary and master units, output is done with respect to the master unit.

CONTROL STATEMENTS

Control statements may be input from the following devices:

- Teletypewriter
- Paper tape
- Cards
- Magnetic tape

At the start of execution, the standard input unit is used as the control statement input unit. A listing of all control statements input appears on the standard list device.

*L STATEMENT

The following is the control statement format.

*L, lu_1, lu_2, lu_3

Where: lu_1 is the unit containing the update binaries or input unit 1 (binary unit).

lu_2 is the old master unit or input unit 2 (master unit).

lu_3 is the unit to receive new output (new unit).

All logical unit numbers used must be the same as for MSOS.

*I STATEMENT

*I,n
I,n,

This statement allows binary or ASCII input on unit lu_1 to be inserted after the program in position n on the master tape. If an *I, n, * statement is used, the binary inserted is the same as the last one specified on the previous *I control statement.

*D STATEMENT

*D, n
*D, m, n

This statement allows the binaries in positions m to n to be deleted from the old master unit lu_2. The deleted positions do not appear in the output.

*R STATEMENT

*R, n
*R, n, *

This statement allows replacement of modules on the old master unit lu_2 with the binary or ASCII input on unit lu_1. The n parameter defines the position on unit lu_2 and an * parameter signals replacement to be made with the last binary used in a preceding control statement.

*S STATEMENT

*S, a, m
*S, a, m, n

This statement selects a module or modules from unit a to be output. The first unit to be output is m and the last unit output is n. The a parameter has one of the following forms:

B	lu_1 unit specified on an *L control statement
M	lu_2 unit specified on an *L control statement

*C STATEMENT

This statement instructs SETUP to list binary and/or ASCII information from unit lu_2, the old master. Each unit of information listed has a positional number assigned to it. The following are some typical examples.

SYSTEM TAPE INSTALL

```
1   *YM, LIBEDT
2   *YM, LOADSD
.       .
.       .
.       .
28  *L
29  DECK – SYSDAT
.       .
.       .
.       .
```

FORTRAN TAPE INSTALL

```
1   *K, 16, P8
2   *JOB
3   DECK – FTN
4   DECK – GOA
5   DECK – CFIVOC
.       .
.       .
.       .
```

Four types of units of information can be assigned and used by SETUP.

- Binary control statements
- Relocatable binary data blocks
- Undefinable binary blocks of length less than or equal to 96 decimal words
- ASCII records of length less than or equal to 80 characters

NOTE

Units referenced in the following description as B, M, and N refer to the following as defined under an *L control statement.

lu_1	Binary update
lu_2	Old master unit
lu_3	New output unit

*O STATEMENT

*O, m, n

This directs SETUP to begin otuput. Parameter m is the old master position reference to begin the output and parameter n is the last position to be output. Multiple *O statements can be used; however, output parameters m and n cannot be reused. For example: *O, 1, 5 and *O, 2, 4 are not allowed. Since the input and old master have been stored on disk, SETUP has no capability to rewind in the case of magnetic tape or reread in the case of punch cards or paper tape.

*E STATEMENT

This is the end of control statement set identifier. It allows SETUP to begin executing the control statements. An *E statement must be the last control statement in the set and must be present to continue execution.

INSTALLATION

SETUP is coded in FORTRAN and assembly language and is installed on the program library. The FORTRAN object library is not required for execution of those segments written in FORTRAN.

OPERATION

The following are the operating instructions to call and execute SETUP.

Type *JOB

Type *SETPV4

SETUP is loaded by the loader. Execution begins with a READ on the standard input unit of the control statements. All control statements are read at the same time during program initiation. When execution has finished, control goes back to the job processor.

EXECUTION

Execution proceeds in several paths, depending on unit specification. It is assumed that an *C control statement has been previously executed to produce directory reference for the old master installation tape.

- Control statements are read from the standard input unit.
- If an *L statement defines parameter b as equal to parameter m or parameter b equal to parameters m and n, the update binaries are read in from unit B unitl an end-of-file is read. A message is issued to the comment device to mount ther old installation tape on unit M (same as unit B). Response to this message is a carriage return.
- The device containing the old master installation tape is read and stored on disk.
- If parameter m equals parameter n or parameter b equals parameters m and n, a message to mount the new tape to receive the newly formed install tape is typed on the comment device. Response to this message is a carriage return. The update is made and stored on mass storage and is output when the unit is ready to receive it.

If an *L statement defines all three units to be different, the update is made and output is as follows.

- Control statements read from the standard input unit
- Binary updates read from unit B
- Building and editing performed from unit M
- Outputting of the new tape or deck to unit N

TIME AND STORAGE CONSIDERATIONS

If units B, M, and N are separate units, the update and output can proceed with the minimum amount of mass storage usage and in the minimum amount of time. If unit B equals unit N, the update binaries are read and placed on mass storage. If only two units are available, assignment must be unit B to unit N and not unit M to unit N. In the latter case, both the update binaries and old installation tape must be stored, thus resulting in excessive mass storage allocation and updating time.

If the control deck contains an *S statement, all the binaries on units B and M must be placed on mass storage. An *S statement must not be mixed in the same execution with *I, *R, and *D statements.

Time and storage minimums are achieved through use of *I, *R, and *D statements during executions where B, M, and N are separate high-speed peripherals. Maximum time and storage result when an *S control statement is used during execution where units B, M, and N are the same physical units.

SETUP ERROR MESSAGES

Error messages are output on the standard list device. Errors occur in two phases: statement reading and statement execution. All errors are fatal; however, some

errors may be delayed fatal (DF) allowing all statements to be read and diagnosed. Errors occurring in the statement execution phase are immediately fatal (IF) and cause an exit to the job processor (refer to the MSOS Installation Handbook).

SETUP CONSTRAINTS AND LIMITATIONS

The following are constraints and limitations imposed by internal programming techniques and functions:

- The maximum logical unit value is 99 (decimal).
- The maximum number of control statements is 1200.
- The maximum reference entry number is 32767 (see next entry).
- The maximum number of entries that can be stored on mass storage is 1000 for both the B unit and M unit. If entries do not need to be stored on disk, the maximum is 32767.
- All reference numbers refer to the unit defined as M in an *L statement.
- An *C may only appear once, which is immediately after an *L statement.
- An *S statement is not to be sorted.
- An *S statement cannot be mixed in a job with an *I, *R, and *D statements.
- The largest control statement may not exceed 16 characters.
- Control statements that are entered (from other than cards) must have two spaces after the last character.
- An end-of-file on paper tape is considered to be a tape motion failure condition; e.g., the reader runs out of tape.
- The mass storage requirements are 50 sectors, plus one sector for each record stored as a result of unit assignment or control statement selection (*S or *I, m, *).
- Mass storage overflow occurs when access reaches the value of MAXSEC defined by the system.
- All input from nine-track magnetic tape is assumed to be binary information. SETPV4 relies upon a switch mode error to change from ASCII mode to binary mode. A switch error does not occur on a 609 Magnetic Tape. This makes the use of SETPV4 with 609/608 combinations useless.

SAMPLE REPLACEMENT USING SETUP

Figure 14-1 shows a control statement deck and table 14-1 shows sample deck structures.

SAMPLE DUPLICATE REPLACEMENT USING SETUP

Figure 14-2 shows a control statement deck and table 14-2 shows sample deck structures.

TWO INPUT SELECT OPTIONS USING SETUP

A control card deck is shown in figure 14-3 and a sample deck structure is shown in table 14-3.

SKELETON EDITOR (SKED)

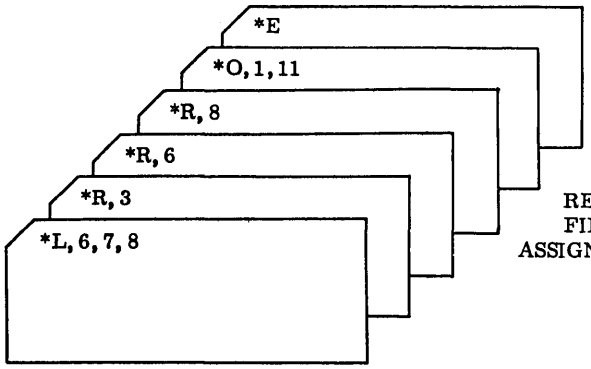
A skeleton is a file that consists of requests to the installation file building program, LIBILD. These requests specify the order and identification of the binary programs that are to be retrieved from a set of library programs and included into the installation file that is being built. The skeleton itself contains no binary programs; it specifies the programs that are to be put in the installation file, which is the output from LIBILD. The skeleton may also contain LIBEDT and system initializer control statements, which will be included in the installation file.

The following are typical skeleton statements:

* B 'SYSDAT'	Include binary program SYSDAT.
* S, END0V4, 7FFF	Assign value 7FFF to entry point name END0V4.
*YM, LIBEDT, 1	Define system directory entry LIBEDT as ordinal 1.
*L,	Load part 0 core-resident programs.
*M	Load system library ordinal.
*K,15	Change standard input to LU 5.
* P	Produce an absolute record.
* END	End of the skeleton file

The purpose of the skeleton editor SKED is to provide a facility for modifying a skeleton to allow changes to an existing system. SKED cannot be used to modify the SYSDAT program, but it can be used to redefine the order and content of the program and system libraries.

To generate a system skeleton, it is necessary to read the system's installation file into SKED via a Build command. To change the skeleton, the user has a number of commands enabling him to add, delete, or change the statements in the file. The modified skeleton may then be output on a convenient medium, such as cards or tape, and used as input to LIBILD.



END-OF-JOB

OUTPUT PROGRAMS 1 THROUGH 11.
REPLACE EIGHTH PROGRAM ON M WITH THE THIRD BINARY FROM B.
REPLACE SIXTH PROGRAM ON M WITH THE SECOND BINARY INPUT FROM B.
REPLACE THIRD PROGRAM ON M WITH THE FIRST BINARY ON B.
ASSIGN LOGICAL UNITS.

Figure 14-1. Control Statement Deck for Sample Replacement Using SETUP

TABLE 14-1. SAMPLE DECK STRUCTURES FOR REPLACEMENT USING SETUP

Tape on Unit B Contains	Tape on Unit M Contains	Tape On Unit N Will Contain
C	A 1	A
F	B 2	B
H	C 3	[C] †
End-of-file	D 4	D
	E 5	E
	F 6	[F]
	G 7	G
	H 8	[H]
	I 9	I
	J 10	J
	K 11	K
	End-of-file	End-of-file

† [ ] Designates programs originating from unit B.

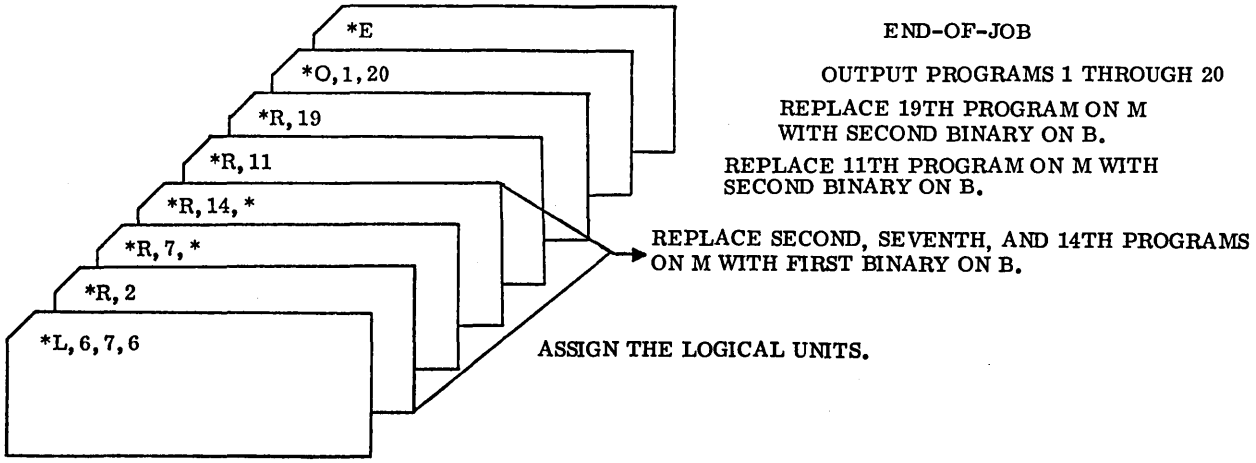


Figure 14-2. Control Statement Deck for Sample Duplicate Replacement Using SETUP

TABLE 14-2. SAMPLE DECK STRUCTURES FOR DUPLICATE REPLACEMENT USING SETUP

Tape On Unit B Contains	Tape On Unit M Contains		Tape On Unit N Will Contain
B	A	1	[A]
K	B	2	[B] †
S	C	3	C
End-of-file	D	4	D
	E	5	E
	F	6	F
	G	7	[B]
	H	8	H
	I	9	I
	J	10	J
	K	11	K
	L	12	L
	M	13	M
	N	14	[B]
	O	15	O

† [ ] Designates programs originating from the B unit.

TABLE 14-2. SAMPLE DECK STRUCTURES FOR DUPLICATE REPLACEMENT USING SETUP (Contd)

Tape On Unit B Contains	Tape On Unit M Contains	Tape On Unit N Will Contain
	P 16	P
	Q 17	Q
	R 18	R
	S 19	[S]†
	T 20	T
	End-of-file	

†[ ]Designates programs originating from the B unit.

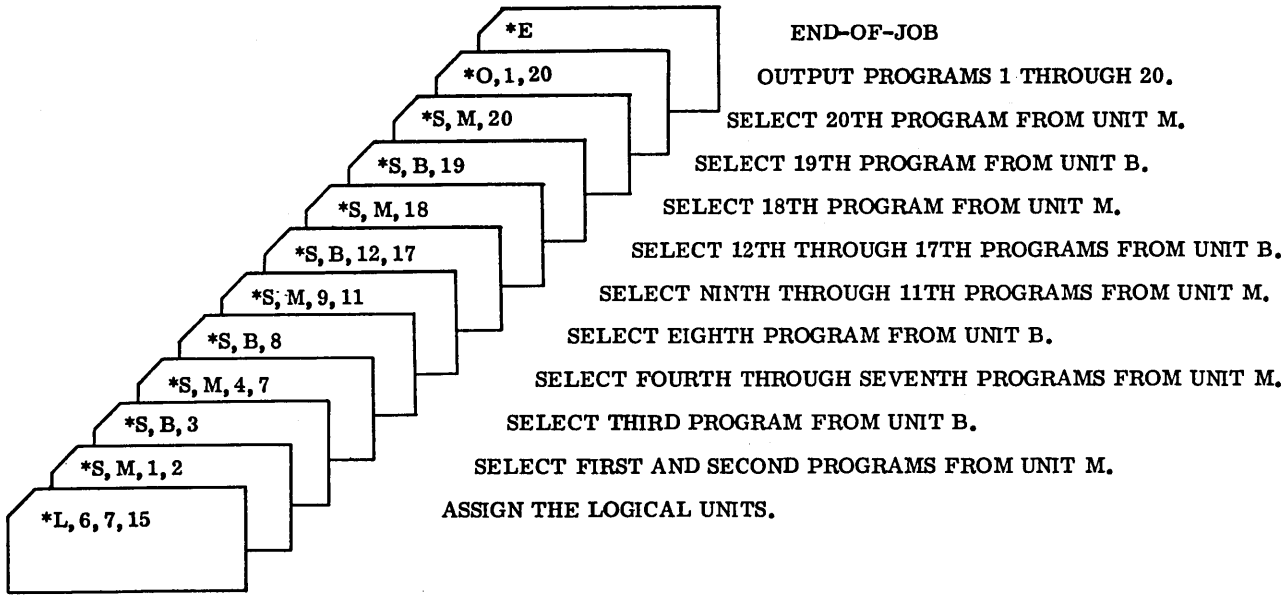


Figure 14-3. Control Statement Deck for Two Input Select Options Using SETUP

TABLE 14-3. SAMPLE DECK STRUCTURE FOR TWO INPUT SELECT OPTIONS USING SETUP

Tape On Unit B Contains	Tape On Unit M Contains	Tape On Unit N Will Contain
A 1	A	A
B 2	B	B
C 3	C	C †
D 4	D	D
E 5	E	E
F 6	F	F
G 7	G	G
H 8	H	H
I 9	I	I
J 10	J	J
K 11	K	K
L 12	L	L †
M 13	M	M
N 14	N	N
O 15	O	O
P 16	P	P
Q 17	Q	Q
R 18	R	R
S 19	S	S
T 20	T	T
End-of-file	End-of-file	

† Designates programs originating from unit B.

NOTE

See the Skeleton Editor section in this manual for a description of the skeleton statements.

In addition to the modification commands, SKED has the facility to list all or part of the skeleton, resequence the record numbers, and allow tape motion control.

SKED runs under the control of the job processor and can be brought into execution by entering an *SKED command. SKED identifies itself by typing SKED IN on the console, followed by NEXT (SKED types NEXT whenever it is ready to receive a command). The user enters a valid command, followed by a carriage return. A command consists of a unique command name followed by any necessary arguments. The command name may be abbreviated to as few letters as will keep the name unique; it may not contain more letters than the forms given in the list below. No embedded blanks are allowed in the command, but commas are required to separate arguments. The following are the editing commands.

LIST	Type a brief description of the valid commands.
COMAND,LU	Change the command input device to LU.
BUILD,LU	Read the installation file from device LU and build the skeleton file in the scratch area.
LOAD,LU	Read the skeleton file from device LU and transfer it to the scratch area.
CATLOG,N1,N2	List records numbered N1 through N2 from the skeleton file.
DELETE,N1,N2	Delete records N1 through N2 from the skeleton file.
INSERT,m,LU	Read new skeleton records from LU and insert in the file immediately following record number n.
DUMP,LU	Write the skeleton file onto device LU.
CHANGE,ILU1,LU2	Find all *K records that specify LU1 as the input device and change LU1 to LU2.
EXIT	Exit from SKED and return control to the job processor or respond to the NEXT statement with a carriage return.

The tape motion commands have the same formats as those in DEBUG. Pseudo tape motion of the skeleton file is accomplished by specifying SK as the logical unit. These commands are:

REW,LU	Rewind LU.
UNL,LU	Unload LU.
ADF,LU,n	Advance n files on LU.
BSF,LU,n	Backspace n files on LU.
ADR,LU,n	Advance n records on LU.
BSR,LU,n	Backspace n records on LU.
WEF,LU,n	Write n file marks on LU.

The following are further descriptions of some of the editing commands:

- COMAND – If Lu is a device other than the standard input comment device, the comments are output on the standard list output device.

- BUILD – When an end-of-file condition is detected, the user is asked:

  ANY MORE INPUT. ENTER LU

  If there is more input, type the logical unit number where the information will be read from, followed by a carriage return. If there is no more input, enter a carriage return.

- LOAD – The load command functions in the same way as BUILD.

- CATLOG – The three forms of the CATLOG command are as follows.

  -CATLOG – Resequence and list the entire skeleton file.
  -CATLOG,n – List the record number n of the file.
  -CATLOG,N1,N2 – List records numbered N1 through N2.

  The only way to resequence the file, with the exception of the DUMP command, is by the simple command CATLOG. Prior to resequencing, there are gaps in the sequence numbers where records have been deleted; any inserted record appears in its proper position, but without sequence numbers.

- DELETE – No record may be referenced that has been deleted or inserted since the file was last resequenced.

  A maximum of 500 record deletions is allowed before the file must be resequenced. When the number of deletions exceeds 500 on a certain command, the delete command is ignored and the user receives a message asking him to resequence the file.

  The last record in the file is the *END record and may not be deleted.

  The command has two forms:

  -DELETE,N1 – Delete record number N1.
  -DELETE,N1,N2 – Delete records N1 through N2, where N1 ≤ N2.

- INSERT – No record may be referred to which has been deleted or inserted since the file was last resequenced. No insertion may be made after the last record.

- DUMP – The skeleton is dumped onto the specified device with an end-of-file written at the end. The file is automatically resequenced and listed after the dump is complete.

ERROR CONDITIONS AND MESSAGES

If a device failure occurs, the appropriate standard MSOS device failure message is printed on the console. This condition commonly occurs with the device being used to build or load the skeleton file (e.g., the card reader has read all the cards without detecting an end-of-file). When failure is not an error, respond to the ACTION query with a CU, which has the same effect as an end-of-file. The program then continues. In this case, the user is asked if there is any more input.

The following are the SKED error messages:

INVALID COMMAND

The command name was not valid.

ERROR IN COMMAND FORMAT

An error was made in the format of the command statement (e.g., omitted argument, omitted comma, etc.).

COMMAND NAME NOT UNIQUE

Not enough letters were included in the command name to specify it uniquely.

LU NOT LEGAL FOR COMMANDS

A check is made for a valid device type for any commands requiring an LU.

SKELETON NOT LOADED

An operation on the skeleton was attempted before the skeleton was loaded or built. These operations are CATLOG, INSERT, DELETE, DUMP, CHANGE.

RECORD NUMBER IS ZERO

A record number of zero was entered as an argument.

INVALID CHARACTER IN NUMBER

A nondecimal character was included in a record number argument.

INVALID RECORD NUMBER

The record number was outside the range of the skeleton file or the second number in the argument list was less than the first.

RANGE CONTAINS NUMBER ALREADY DELETED

A reference was made to a record that was deleted since the file was last resequenced.

RECORDS HAVE BEEN PREVIOUSLY DELETED

A CATLOG command was entered in which the range of numbers requested have all been deleted.

NO INSERTION RECORDS AT SPECIFIED LU.

The device specified did not contain any insertion records.

RECORDS NOT DELETED PLEASE RESEQUENCE SKELETON

An attempt was made to delete more than 500 records since the file was last resequenced.

RESPONSE MUST BE LU(CR) OR (CR)

An invalid response was given to

ANY MORE INPUT. ENTER LU

ENTER COMMANDS ON INPUT COMMENT DEVICE

This message will appear when one of the preceding error messages has occurred and the command input device was not the console.

LIBRARY BUILDER (LIBILD)

The library and installation file builder (LIBILD) provides the following capabilities:

- Merges input libraries of relocatable binary programs into a single output library and discards duplicated programs
- Produces an installation file suitable for building a system via the system initializer or LIBEDT
- Conversational control statements
- Batch control statements
- Absolutized file input and output to installation file; e.g., MACSKL, MACROS
- Input and output devices fully selectable via logical unit designations
- Substantial recovery features
- Diagnostics in English
- Prompts messages and pauses at appropriate times to allow the operator to mount tapes, etc.

SUMMARY OF OPERATION PHASES

- Control statements – Required
- Library input – Required
- Library output – Optional
- Definitions input – Optional
- Skeleton input and installation output – Optional

DEFINITION OF TERMS

Absolute file

A set of binary records (whose first record is neither a NAM block nor an asterisk) preceded by a record of the form:

```
*N,xxxxxx,,,B
```

and terminated by a record having an asterisk (*) as the first character. For example, if the macro skeleton is on an input library it must take the following form.

```
*N,MACSKL,,,B
.
.                  } absolute binary records
.
*                  (anything)
```

The name given to the absolute file is that of the name field of the *N record (MACSKL in this example).

Control Statement

The operator's response to a query made by LIBILD. The response may be entered via the standard input comment device or another device specified by the operator.

Definitions

A set of ASCII records, each of which has an asterisk as the first character, that terminates with an *END or end-of-file indication or an I/O error answered by CU. A definition group begins with an *DEF record, contains only valid skeleton records, and ends with an *TER record. Definition records have the following format:

```
*DEF x
```

This record directs LIBILD to set up an internal directory entry identified by the single character x, which must be unique for each *DEF reocrd. Subsequent records are read until an *TER record is detected; those records are then referred to as group x. This process is analogous to the CYBER 18/1700 macro assembler definition. Every record in the group must begin with an asterisk.

Example:

```
*DEF    F
*B      'FILMGR'        '39795252'
*B      'RTNSPC'        'MSOS 5.0'
*B      'DEFFIL'        'MSOS 5.0'
*B      'RELFIL'        'MSOS 5.0'
*B      'RTVSEQ'        'MSOS 5.0 VO2'
*B      'STOSEQ'        'MSOS 5.0 VO3'
*TER
```

Note that the identification field may contain any characters, with the exception of single quote marks which are used as delimiters.

Duplicate program

A relocatable binary program or subprogram whose NAM block contains a name and identification equal to a previously input program or subprogram. In the case of absolute files, only the name is used as a basis of comparison.

Identification

The information beginning in column 25 of the NAM block; it is scanned until six successive blank columns are encountered, at which point the end of the identification is assumed.

If two programs have identical identification fields with one program having additional identification information as in the following example, the user must ensure that the identification field of each *B record contains sufficient data to make it unique. For example, two program identification fields appear in the following order:

```
NAM MIPRO       DECK-ID A26 MSOS 5.0 1700
                NO. 1
NAM MIPRO       DECK-ID A26 MSOS 5.0
```

Each *B record must be unique. If

```
*B  'MIPRO' ' DECK-ID A26 MSOS 5.0'
```

is input, the first program matches up to the point at which the ' mark is encountered. Thus, the first program is loaded even though additional identification exists on the NAM block because the ' mark terminates the search.

In order to obtain the second program, the *B record must then appear as

```
*B  'MIPRO' ' DECK-ID A26 MSOS 5.0 '
```

By adding spaces to the end of the identification information before the ' mark, this identification information is unique and causes the proper program selection.

Leading spaces in the identification field of a NAM block are ignored unless the entire field is spaces. A *B record which has identification information of all spaces matches correctly with a NAM record which has a blank identification field. For example, the record

```
*B  'NAME01'    '   '
```

matches with the specific program named NAME01 with a blank identification field.

```
*B  'NAME01'
```

matches with the first occurrence of the program named NAME01, regardless of the identification field information.

Input library

A set of relocatable binary programs and subprograms that are terminated by an *END record or an end-of-file indicator, or an I/O error answered by a CU. A library can have any number of system initializer and LIBEDT control statements; thus, an existing installation tape can be used as a library. Absolute files such as MACSKL can be a part of a library.

Installation file

When produced by LIBILD, the installation file is the set of relocatable programs, subprograms, and absolute files specified by the *B records in the skeleton, plus all other records in the skeleton that have no meaning to LIBILD.

Usually the installation file is suitable input to the system initializer or LIBEDT.

Output library

Produced from one or more input libraries; a set of relocatable binary programs and subprograms terminated by an *END record.

Skeleton

A set of ASCII records, each of which has an asterisk as the first character, that terminates by an *END or end-of-file indication or an I/O error answered by CU. These can be system initializer and LIBEDT control statements, LIBILD control statements, or anything else supported by MSOS. The skeleton defines the logical sequence and content of information written to the installation file. The following record formats are recognized as LIBILD statements:

*B 'aaaaaa' 'bbbbbb'

The *B statement directs LIBILD to retrieve a relocatable binary program, subprogram, or absolute file and to write the entire program on the installation file. The name of the program is specified by a one- to six-character name, enclosed by single quote marks. The identification field provides the capability to differentiate between programs having the same name. Leaving the identification field blank (without even quote marks) causes the first copy of several copies or the only copy of a program to be retrieved.

The *B record for an absolute file should have a blank identification field. The following is an example of an *B record:

```
*B 'JOBENT'    'DECK-ID 031 MSOS 5.0'
```

Quote marks may begin anywhere after column 2; embedded blanks are significant.

*WEF

This record directs LIBILD to write an end-of-file indication of the installation file.

*USE A

This record is the counterpart of a CYBER 18/1700 Macro Assembler call and is used to call out or specify that the records grouped under a previous *DEF A record are to be inserted in the skeleton at this point. The *USE record may be nested to a depth of six levels.

Example:

```
*DEF   A
*B     'PROGRM'      } Defines symbol A
*TER
*DEF   B
*USE   A†
*B     'PROGRA'      } Defines symbol B
*TER
```

†Insert definition for A, B, or C.

```
*DEF   C
*USE   B†            } Defines symbol C
*TER
*DEF   D
*USE   A
*USE   B             } Defines symbol D
*USE   C†
*TER
```

When specifying an *USE record, the *DEF and *TER of the corresponding symbol are not inserted in the skeleton.

*END

This record signifies the end of the skeleton. It is written to the installation file.

*character string

When the character string is not identically equal to B, WEF, DEF, TER, or USE, the control statement has no meaning for LIBILD. The record is written to the installation file.

Example:

```
*K,15,P8
*LIBEDT
*T
*Z
*U
```

OPERATION

1. LIBILD resides in the program library and is executed by the job processor in response to the following control statement:

   *LIBILD

2. The program always requests its first control statement from the standard comment input device as follows:

   CONTROL LU =

   This query is for determining the logical unit from which the program reads subsequent control statements, such as a card reader. If the operator types only a carriage return, the queries and control statements remain on the standard comment device. If a logical unit number is entered, each query is printed on the standard list device prior to reading the control statement, which is also printed.

   A negative reply to any query is either a carriage return on the comment device or a card image having a blank in column 1, whichever is appropriate. If a card is used, the entire card is printed on the list device following the query message and the characters following the first blank are ignored.

   All positive replies are followed by a carriage return or a blank.

3. The next statement is used to specify the logical unit from which definitions will be read.

   DEFS LU =

4. This is followed by the statement which specifies the logical unit on which the installation file is to be written:

   INSTALL LU =

5. The next statement specifies the output library logical unit:

   NEWLIB LU =

6. Up to nine input libraries may now be specified. The first query of this type is:

   LIB 01 LU =

   A logical unit number should be typed at this point. The next query is:

   LIB 01 LU =

   On this and subsequent replies, the negative reply signifies that there are no more library logical units. The number following LIB is just a library number counter.

7. The next query is produced when the library logical units have been specified.

   SKELETON LU =

   Following this reply, the first library is read.

   During the library input phase, each program name and identification field is printed on the list device.

DIAGNOSTICS AND MESSAGES

During the control statement input phase, the logical unit numbers are checked for legality. If the number is out of the logical unit range or is indecipherable, this message is produced:

INVALID LU (1)

If the number specifies a device that is incompatible with the function to be performed on that device, this message is produced:

INVALID CLASS CODE (2)

If the control statement device is the standard comment device and one of the above diagnostics is produced, the routine repeats the query; otherwise, the routine exits to the job processor.

Library Input Phase

During the library input phase the following messages can appear on the list device:

LAST DECK REJECTED - NOT UNIQUE (3)

This means that the immediately previous program had the same name and identification as another program. The program that is initially loaded, having a specific name and identification, is retained.

LAST DECK REJECTED - NO XFR RECORD (4)

This message is produced in conjunction with another message on the comment device when the operator chooses to continue library input rather than aborting the job.

During the library input phase the following messages can appear on the comment device:

NAM RECORD NOT 1ST RECORD OF DECK. (5)
TYPE 1, CR TO TERMINATE EXECUTION.
TYPE 2, CR TO PROCEED TO SUBSEQUENT LIBRARY OR SKELETON.
TYPE 3, CR TO CONTINUE ON WITH CURRENT LIBRARY.

This means that the anticipated relocatable binary deck did not have a NAM block format. The message tells the operator to type 1, 2, or 3, followed by a carriage return and the noted action will occur.

1 Exit to the Job Processor.

2 Ignore the remainder of the current library (just as though it has been completely processed).

3 Slew over the records until either a record having an asterisk as the first character or a NAM block is detected; then resume at that point as if no error had occurred.

XFR RECORD MISSING FOR LAST PGM LISTED. (6)
PGM DELETED.
TYPE 1, CR TO TERMINATE EXECUTION.
TYPE 2, CR TO PROCEED TO SUBSEQUENT LIBRARY OR SKELETON.
TYPE 3, CR TO CONTINUE ON WITH CURRENT LIBRARY.

This means that a relocatable binary deck was being input and after the initial NAM block, either a record having an asterisk as the first character or another NAM block was detected prior to finding a

XFR block. The instructions to the operator and the corresponding action taken are the same as the previous statement.

TOO MANY BINARY DECKS LOADED. CHANGE LIMIT AND RECOMPILE. (7)

LIBILD can handle a fixed number of unique program names. (See restrictions and limitations.) This message means that the fixed number has been exceeded and that, unless fewer programs are present on the libraries, LIBILD itself must be modified and reinstalled.

LOAD LIBRARY INPUT xx ON LU yy. CR WHEN READY. (8)

This is not an error message. If more than one library was specified during the control statement phase, this message appears after the first library and each subsequent library is input. The library count is xx and the logical unit number is yy; all such numbers in this message occur in the order they were defined during the control statement phase. When this message is printed, the operator may change tapes, reload the card reader, etc. When the desired library is finally on the device, depressing the carriage return key will cause that library to be input.

Library Output Phase

During the library output phase the following message can appear on the comment device:

LOAD OUTPUT LIBRARY. CR WHEN READY. (9)

This is not an error message. It always appears when an output library option has been selected. When the device is ready, depressing the carriage return key causes the relocatable binary programs to be written to the device.

During the library output phase, each program name and identification is printed on the list device and this error message can appear:

CHECKSUM ERROR NOTED IN LAST PROGRAM. (10)

When a program is input from a library, a checksum for each record is generated and the record and its checksum are transferred to mass storage. When that program is retrieved during the library or installation file output phase, the checksum is recomputed and compared with the original checksum. Differences result in the above messages and cause that program's records to be terminated, resulting in an incomplete relocatable binary on the file.

Definitions Input Phase

During the definitions input phase this message appears on the comment device:

LOAD DEFS, CR WHEN READY (11)

This is not an error message. The purpose of this message is to allow the operator time to load and ready the device. Depressing the carriage return key causes the definitions to be input.

During the definitions input phase, each definition record is printed on the list device. One of the following error messages can be printed after the definition record is printed.

BAD *DEF RECORD. NO IDENT CHARACTER. (12)

See the Definition of Terms section.

BAD *DEF RECORD. IDENT CHAR ALREADY USED. IGNORED (13)

See the Definition of Terms section.

INVALID DEFINITION RECORD. IGNORED. (14)

The first record of a definition group was not an *DEF. Records are slewed until an *DEF is encountered.

NO DEFINITIONS WERE SUCCESSFULLY LOADED. (15)

This message is self-explanatory.

TOO MANY DEFINITION SETS. IGNORED. (16)

There is a limit to the number of definitions that can be input. This message means that the limit has been exceeded and subsequent records are slewed over until the *END record is encountered. (See Restrictions and Limitations.)

Skeleton Input Phase

During the skeleton input phase this message always appears on the comment device:

LOAD SKEL/INSTAL, CR WHEN READY (17)

This is not an error message. The purpose of this message is to allow the operator time to load and ready both the skeleton input and the installation output devices. Depressing the carriage return causes the skeleton records to be input and the installation records to be output concurrently.

During the skeleton input and installation file output phase, skeleton records are normally printed on the list device under one of two conditions. If the record has no meaning for LIBILD, it is written to the installation file. If the record has meaning for LIBILD, the appropriate substitution or action is performed. (See Definition of Terms.)

During the skeleton input and installation output phase, one of the following error messages can occur after the printing of the skeleton record:

PROGRAM SPECIFIED BY THIS RECORD NOT FOUND. (18)

This means that the name on the *B record did not match the name of any program read during the library input phase. Nothing is written to the installation file for this record.

PROGRAM HAVING THIS ID INFO NOT FOUND. (19)

The name on the *B record matched at least one name of a program read during the library input phase, but the identification field on the *B record did not match the identification field of any of those same programs. Nothing is written to the installation file for this record.

MORE THAN ONE PROGRAM HAS THIS NAME (NO ID INFO.) (20)

The name on the *B record matched more than one of the program names read during the library input phase, but no identification was specified on the *B record. The first program loaded during library input having the specified name is written to the installation file.

CHECKSUM ERROR NOTED IN LAST PROGRAM. (21)

See Library Output phase.

ILLEGAL CHARACTER STARTS IDENT FIELD. (22)

On an *B record the identification field must start with a single quote mark. Nothing is written to the installation file for this record.

ILLEGAL IDENT FIELD. RECORD IGNORED. (23)

The identification field on an *B record was not terminated by a single quote mark prior to column 69. Nothing is written to the installation file for this record.

ILLEGAL *B RECORD. RECORD IGNORED. (24)

No leading or trailing single quote for the name field detected prior to column 73. Nothing is written to the installation file for this record.

NULL PROGRAM NAME. RECORD IGNORED. (25)

The name field on an *B record consisted of only two single quote marks. Nothing is written to the installation file for this record.

PROGRAM NAME TOO LONG. RECORD IGNORED. (26)

The name field on an *B reocrd had more than six nonblank characters between the single quote marks. Nothing is written to the installation file for this record.

NO DEFINITIONS ARE STORED. RECORD IGNORED. (27)

An *USE record was encountered and no definitions are present. Nothing is written to the installation file for this record.

INVALID *USE RECORD. IDENT FIELD. RECORD IGNORED. (28)

No nonblank character was detected prior to column 69. Nothing is written to the installation file for this record.

INVALID *USE RECORD. MAX EMBEDDED LEVEL IS 6. RECORD IGNORED. (29)

See Definition of Terms.

INVALID *USE RECORD. REQUESTED SET IS IN USE. RECORD IGNORED. (30)

An embedded *USE record was encountered having an identification field identical to that of the current *USE record. This is an infinitely recursive condition.

What is written to the installation file is dependent upon the logic flow of the *USE expansion. Refer to the printer output to determine what was written.

END FILE MARK WRITTEN (31)

This is not an error message. It appears whenever an *WEF record is encountered. (see Definition of Terms.)

FATAL PROGRAM ERROR. RUN KILLED. (31a)

The sector number in the directory of program names points to a sector whose first word is zero. This should be the length of the record. This indicates some error has occurred in defining the sector address of the program in the program directory, or the program data has been written incorrectly.

At the completion of the skeleton input phase, the following message is output on the comment device:

```
LIBRARY BUILD COMPLETE                    (32)
TYPE *Z TO TERMINATE OR
TYPE *C TO CONTINUE WITH CURRENT
    SKELETON AND/OR
OUTPUT LIBRARY LU'S
```

If the operator types *Z followed by a carriage return, LIBILD exits to the job processor. If he types *C, the routine is recycled to the point at which the skeleton and installation file devices are loaded and set ready. In other words, a different skeleton and installation file may be loaded but only on the same devices used by the first skeleton and installation file; the same set of library programs is retained. This feature could be used after skeleton errors are detected and corrected.

There is no program limit to the number of times this feature may be exercised.

RECOVERY FROM ERRORS

If diagnostic message 5 is output during the library input phase, there may be a transient hardware problem (e.g., tape positioning, card feed problems) and the operator should manually backspace the library and try again by typing 3 in response to the message. The card reader is backspaced by refeeding the card.

Magnetic tape backspacing is more complex. If the operator has the time, he may rewind the tape and reprocess it, obtaining diagnostic message 3 for every program successfully loaded on the previous pass of that tape.

If the library input is already very large, a slightly more complicated but faster process can be used: Manually backspace the tape a few feet. The tape will now be positioned in the middle of a record, at the beginning of a record that is not a NAM block or a record beginning with an asterisk, or at a NAM or asterisk record. Type 3 is followed by carriage return and a record will be read.

If the tape position was at an asterisk or NAM record, the input library processing proceeds with reject message 3.

If the tape was positioned elsewhere, message 5 is output again.

Typing 3 causes LIBILD to slew over records until an asterisk or NAM record is encountered. Processing this input library proceeds with reject message 3.

If skeleton format errors are detected or certain skeleton records are out of order or missing during the skeleton input phase, the operator can correct or rearrange the skeleton deck and try again after message 32 is output.

If, during the control statement phase, the operator accidentally specifies more input libraries than he really has and all of the libraries have been input, in lieu of starting over he can do the following.

1. When message 8 occurs for the nonexistent library, verify that the device is not ready and type carriage return. The operating system will output a failure message for the device.

2. Type CU.

3. LIBILD defaults an *END at this point and proceeds as though a library had been input.

Perform these three steps for each nonexistent input library. The following is a typical LIBILD operation:

```
                                        Comments

*LIBILD
CONTROL LU   =                          Control stays at
DEFS LU      = 5                        comment device
INSTALL LU   = 6
NEWLIB LU    = 7                        All input libraries
LIB 01 LU    = 6                        merge to this device

LIB 02 LU    = 5
LIB 03 LU    = 7
LIB 04 LU    = 14
LIB 05 LU    = 5
LIB 06 LU    =                          No more libraries.
SKELETON LU  = 5                        Also ensure library 01
                                        on LU 06 is ready.

[            ]                          At this point library 01
[            ]                          is input on LU 06.

LOAD LIBRARY INPUT
02 ON LU 05. CR
WHEN READY.

[            ]                          Library 02 is input.
[            ]

LOAD LIBRARY INPUT
03 ON LU 07. CR
WHEN READY.

[            ]                          Library 03 is input.
[            ]

LOAD LIBRARY INPUT
04 ON LU 14. CR
WHEN READY.

[            ]                          Library 04 is input.
[            ]
```

LOAD LIBRARY INPUT 05 ON LU 05. CR WHEN READY. [ ]	Library 05 is input.
LOAD OUTPUT LIBRARY. CR WHEN READY. [ ]	At this point all programs from libraries 01 – 05 are output to LU 07.
LOAD DEFS. CR WHEN READY. [ ]	Definitions are input from LU 05.
LOAD SKEL/INSTAL. CR WHEN READY. [ ]	Skeleton is input from LU 05 and the installation file is written on LU 06.
LIBRARY BUILD COMPLETE TYPE *Z TO TERMINATE OR TYPE *C TO CONTINUE WITH CURRENT SKELETON AND/OR OUTPUT LIBRARY LU'S [C]†	Operator wants to continue.
LOAD SKEL/INSTAL. CR WHEN READY. [ ]	Another skeleton and installation file are processed.
LIBRARY BUILD COMPLETE TYPE *Z TO TERMIN- ATE OR TYPE *C TO CONTINUE WITH CURRENT SKELETON AND/OR OUTPUT LIBRARY LU'S [*Z] J [*Z]	LIBILD is terminated.

RESTRICTIONS AND LIMITATIONS

- The maximum number of unique program/identification combinations is 1024.
- The maximum number of input libraries is nine.
- The maximim number of definition groups or sets is 20.
- The maximum number of recursive levels in a definition is six.
- There is no limit to the maximum number of times a given program name/identification can appear on an *B record.
- The maximum number of times a given program name can have a different identification is 1024.
- There is no limit to the maximum number of times the same program name/identification is rejected if it is encountered more than once.
- There is no limit to the number of records in a skeleton.
- There is no limit to the number of skeleton input.
- Changing any of the above limits requires a nontrivial program modification.

SPECIAL NOTES

The name and identification information of a binary program are obtained by SKED during the BUILD operation from the NAM record of that program. It may be desired to exclude certain information from the identification field of the NAM card (columns 25 through 68) in the identification field of the skeleton record that is generated, because of possible problems that may occur later when running the library builder program, LIBILD.

When replacing a program on one input library with a program on another input library, LIBILD recognizes the program to be replaced by comparing the programs' names and identification fields in their respective NAM records. If any information is different, such as a date, LIBILD treats the two programs as though they were unique and both programs will be included in the installation file.

General case: Any information on the NAM card that is not to be included in the identification field of the skeleton record should follow the information that is to be included by at least four blank columns.

Special case: If none of the information in the identification field is to be included, at least eight blank columns must be left at the beginning of the identification field (i.e., the identification information on the card should start in or after column 33 on the NAM card).

MACRO LIBRARY BUILDER (LIBMAC)

LIBMAC is the macro library preparation routine. Input to LIBMAC is a set of source macro definitions. The set of macro definitions is terminated by ENDMAC, starting in column 1 of the source image.

† [ ] represents operator entry.

The procedure to execute this program is:

```
*JOB
J
*K,Ilu,Plu
J
*LIBMAC
```

Where: I assigns the logical unit of input.

P assigns the logical unit of output.

Output from this routine is in the following form:

- Macro skeletons
- End-of-file
- Macro directory
- End-of-file

The following control statements can be used to put MACSKL and MACROS on the program library:

```
*JOB
J
*LIBEDT
    LIB
    IN
*K,Ilu
*N,MACSKL, , , B
*N,MACROS, , , B
```

LIST SYSTEM LOGICAL UNITS (LULIST)

The LULIST program is a utility function that lists the system logical units.

A typical operation is:

Operator Action	System Output		Comments
*LULIST	ON LIST DEVICE	ON COMMENT DEVICE	
	E10		Call program into execution
	LOG1A		
		E	Loader message Link to CREP table
	(Logical unit listing)		
	J		End of operation

LIST PROGRAM NAMES (LISTR)

LISTR lists the name and record length of all programs on a binary tape. It is used primarily for validating and listing installation tapes whenever programs have been added or replaced. This program should be installed on the 1700 MSOS program library using an *L, entry point name function of LIBEDT (refer to the MSOS Installation Handbook).

The procedure to execute this program is:

```
*JOB
J
*K,Ilu,Llu
*LISTR
```

Where: I assigns the logical unit for the binary programs to be listed.

L assigns the logical unit for the printout.

Examples for the standard printer and the FORTRAN printer are shown in figures 14-4 and 14-5.

SORT OPERANDS (OPSORT)

The operand sort program (OPSORT) cross references CYBER 18/1700 assembly language operands. The program operates on the list output of the 1700 Macro Assembler. It reads the list output from the standard input device (cards, paper tape, magnetic tape). Each record is sorted by operand; an alphabetic listing of all operands is given on the standard list output device. The listing includes the source card number and operation code for each operand.

If any macros are present in the program, they are printed along with their location numbers prior to output of sorted operands.

OPSORT is a debugging aid to assist in isolating program errors. References to all labels are listed alphabetically, thus allowing the user to identify all instructions and their locations that affect or use the code at that point. OPSORT can be used when changing or inserting labels to ensure that duplication does not occur.

The procedure to execute this program is:

```
*JOB
J
*K,Ilu,Llu
J
*OPSORT
```

Where: I assigns the logical unit of the list tape.

L assigns the logical unit of the list device.

The following is an example of the output:

```
          NAM       MINT
0040      SCHDLE    M11A,3          Down to level 3
0115      SCHDLE    (JPCHGE),       Yes-Schedule
LLS       0077      8
LDA*      0192      ALVST),Q
STA*      0196      ALVST),Q
LDA-      0194      HICORE
EQU       0033      HICORE F6
EXT       0024      JBCNCL
```

```
1
+   001    *S,SYSMON,$3-38
+                                          0001 RECORDS        0001 TOTAL RECORDS..

+   0002   *S,SYSDAY,$3136
+                                          0001 RECORDS        0002 TOTAL RECORDS..

+   0003   *S,SYSYER,$3734
+                                          0001 RECORDS        0003 TOTAL RECORDS..

+   0004   *S,SYSLVL,$3832
+                                          0001 RECORDS        0004 TOTAL RECORDS..

+   0005   *V
+                                          0001 RECORDS        0005 TOTAL RECORDS.

+   0006   *V        1700 MASS STORAGE O
+                                          0001 RECORDS        0006 TOTAL RECORDS..

+   0007   *V
+                                          0001 RECORDS        0007 TOTAL RECORDS..
```

Figure 14-4. Standard Printer Example

```
0001    *S,SYSMON,$3038                    0001 RECORDS        0001 TOTAL RECORDS..
0002    *S,SYSDAY,$3136                    0001 RECORDS        0002 TOTAL RECORDS..
0003    *S,SYSYER,$3734                    0001 RECORDS        0003 TOTAL RECORDS..
0004    *S,SYSLVL,$3832                    0001 RECORDS        0004 TOTAL RECORDS..
0005    *V                                 0001 RECORDS        0005 TOTAL RECORDS..
0006    *V        1700 MASS STORAGE O      0001 RECORDS        0006 TOTAL RECORDS..
0007    *V                                 0001 RECORDS        0007 TOTAL RECORDS..
```

Figure 14-5. FORTRAN Printer Example

OPSORT can be used to sort a single program or multiple programs prior to returning to the job processor. An exit from OPSORT occurs when the SKIP switch is set. Therefore, when sorting a single program, set the SKIP switch immediately. On a multiple program sort, set the SKIP switch during execution of the last program to be sorted.

LIST AND SORT PROGRAMS (EESORT)

EESORT is a utility program that processes relocatable binary programs and prepares a listing of information that is unique to those programs. EESORT resides in the program library on mass memory and is executed in unprotected core.

EESORT is called from the program library and placed in execution in the following manner.

EESORT requests the user to specify the LIST or SORT option via the standard comment device. After the user enters a response, EESORT continues reading from the standard input device until an *T is encountered. If the LIST option is specified, EESORT prepares a listing for output on the standard list device, which contains the following information.

- Program name
- Name card comments
- Program length
- Common size
- Data size
- Entry points
- Externals referenced

If the SORT option is specified, EESORT prepares a listing of the program names (sorted alphabetically), the declared entry points, and the programs that reference these entry points as externals.

The information that is to be sorted is recorded within the bounds of unprotected core. If insufficient unprotected core is available, no sort is performed and the following diagnostic is printed:

MEMORY OVERFLOW – NO SORT

If a binary program appears several times within the file being processed, it is stored each time it is encountered.

A sample job deck is shown in figure 14-6.

COSY FORMAT (CYFT)

The COSY format program is used to insert the proper COSY control cards in assembly language programs so that the generated output can be input to build COSY source programs. Input source is from the standard input device (cards, paper tape, magnetic tape). If input is from magnetic tape, the last deck of the input source must have an end-of-file mark.

The COSY format program should be installed on the CYBER 18/1700 MSOS program library using an *L, entry point name function of LIBEDT (refer to CYBER 18/1700 MSOS Installation Handbook).

The procedure to execute this program is:

```
*JOB
J
*K,Ilu,Plu,Llu
J
*CYFT
```

Where: I assigns the logical unit for the source program input.

P assigns where COSY control cards and source programs are to be punched (written).

Example:

```
xxxxxx  DCK/          I,C    (CYFT gen-
                             erated)
xxxxxx  HOL/                 (CYFT gen-
                             erated)
        NAM xxxxxx
        END
        END/                 (CYFT gen-
                             erated)
```

L assigns the logical unit for a listing of COSY DCK/ cards to be listed.

Example:

```
xxxxxx  DCK/          I,C
xxxxxx  DCK/          I,C
```

If the logical unit fails upon completion of input, type in CU. This generates the final END/ for COSY.

COSY PROGRAM

The CYBER 18/1700 COSY program provides a means of compressing information in source decks by replacing three or more blanks on a card with two special ASCII characters.† COSY compresses Hollerith source decks and converts the Hollerith code to ASCII code. The resulting deck, called a COSY deck, is in COSY format (see COSY Library). COSY reduces average deck size by about 60 percent.

A COSY library consists of a group of COSY decks. Each COSY deck is preceded with a COSY deck identifier card and terminated with an end-of-deck character. The COSY library may be written on paper tape, magnetic tape, or punched cards, and is terminated with an END/ card followed by an end-of-file mark.

The COSY program is called from mass storage by typing *COSY and pressing RETURN on the teletypewriter console or with a *COSY punched card. There are no parameters for the teletypewriter call to COSY or for the *COSY card.

A COSY revision deck follows the call to COSY. COSY revision decks (see Revision Deck) allow the user to prepare, revise, or copy COSY decks, and to prepare, update, or copy COSY libraries. COSY may be used with any source language that does not use COSY control statements. COSY output may be in Hollerith or COSY (compressed ASCII) format and may be listed, punched, or sent to a compiler or assembler.

COSY CARDS

COSY revision decks are comprised of COSY control cards and new source cards. There are seven COSY control cards (MRG/, DCK/, CPY/, DEL/, INS/, REM/, and END/) and two deck identifier cards (HOL/ and CSY/). The fields for all COSY control and identifier cards (except DEL/ and INS/) are in the following standard format:

1	8	13		73
deck name	card name	parameters	comments	id

deck name — Columns 1 through 6; the name of a deck in a COSY library that is to be modified or copied. Deck name is used only on DCK/, CPY/, HOL/, and CSY/ cards. The field is blank on all other COSY cards.

card name — Columns 8 through 11; the name of the COSY control card.

parameters — Start in column 13; parameters are terminated by a space.

†In this section, card refers to any single input/output record, and deck refers to a set of cards.

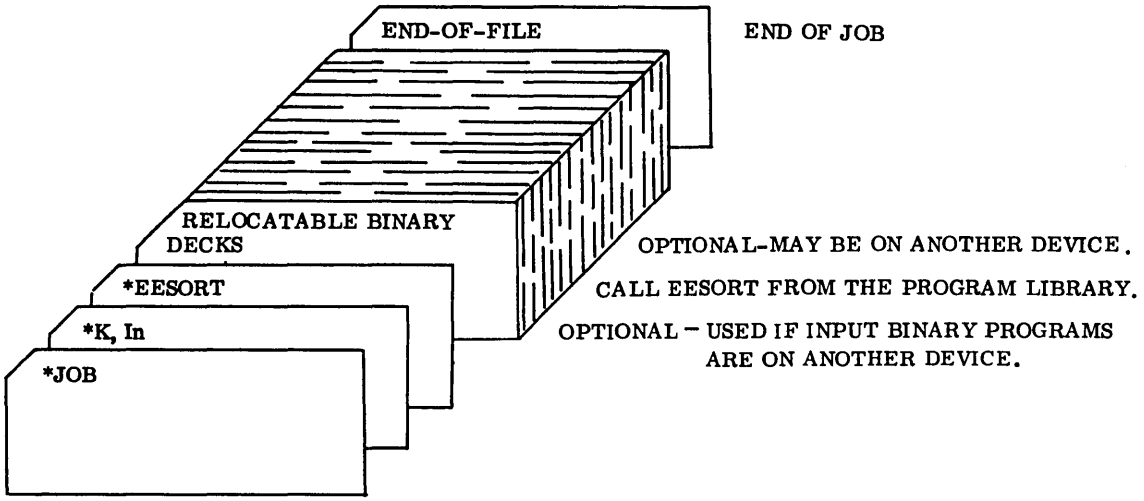


Figure 14-6. EESORT Sample Job Deck

comments — Can start in any column after the terminating space for parameters; comments may run through column 72 and are optional.

id — Columns 73 through 75; a three-character deck name identifier, used only on DCK/, HOL/, and CSY/ cards.

The control card fields for DEL/ and INS/ cards are in the following format.

1	8	13	66	72
	card name	para-meters	comments	change record

The card name, parameter, and comment fields are the same as for the standard card above, except that the comment field ends in column 65. A change record field is added to these cards to add change identification information.

The change record field is a seven-character field (columns 66 through 72) used to identify the type, nature, or date of a change. COSY writes an asterisk in column 73 and the contents of the change record record field in columns 74 through 80 of each new source card following the INS/ or DEL/ card. This provides a means of identifying new or changed source cards when a COSY deck is listed. Adding change record information on an INS/ or DEL/ card is a user option, which is not required input to COSY.

MRG/Card

An MRG/ card directs COSY to merge two revision decks (see Revision Deck).

1	8	13
	MRG/	a, b, c

a,b,c — Specifies actions to be taken – This card directs COSY to merge the revision deck on logical unit a with a revision deck on logical unit b and write a merged revision deck on logical unit c.

If revisions between a and b conflict, revisions from a are used. The conflicting revisions from b are listed with asterisks in columns 2 through 5 on the standard print device and not on unit c.

If either a or b is missing or zero, COSY assumes the decks are on the standard input device. If c is missing or zero, the standard output device is used.

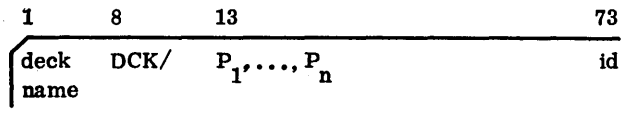
If a and b are the same logical unit, the first revision deck is written onto mass storage and then merged with the second revision deck on the logical unit. Revisions on mass storage have priority if conflicts occur.

The DCK/ card in the merged deck is the DCK/ card from unit a. The merge terminates when the END/ card on both decks is read.

COSY locates a DCK/ card on unit a and searched unit b until the deck names match. If COSY reaches the end of the revision deck on unit b before obtaining a match, it treats all the remaining decks on unit a as new decks and inserts them at the end of the merged deck. If revisions are to be input from different input devices, logical units must be specified on the MRG/ control card.

DCK/ Card

A DCK/ card identifies the COSY or Hollerith deck to be updated or created and specifies the actions to be taken with the new deck.



deck name — Names the COSY or Hollerith deck to be processed

$P_1,\ldots,P_n$ — Specifies the actions to be taken – All parameters are optional, can be in any order, and are separated by commas. Blanks are now allowed within the parameter field. Parameters have the form:

p, or p = lu, or D = deck name

where p is I, C, H, or L, and lu is the logical unit on which input or output occurs. Deck name specifies a new deck name for the COSY output.

I Parameter (Input)

I = lu or I — I specifies the logical unit containing the COSY or Hollerith source deck(s) to be updated or created. If the parameter is absent or just I, COSY assumes the source deck is on the COSY standard input device. (COSY standard devices are set by a *CSY,... statement, see section 9.)

If I = lu is used and lu is the system standard input unit, COSY assumes a new deck is being added to the COSY library. If the first card after the DCK/ card is a source deck identifier, COSY assumes it is a new deck to be added to the COSY library. COSY processes the deck until an END/ card is read. Additional new source decks may follow.

Each new deck nust begin with a source deck identifier card and end with an END/ card. The card following the END/ card must be a DCK/, MRG/, or another END/ card to mark the end of the revision deck.

If the first card after the DCK/ card is not a COSY or Hollerith source deck identifier card, COSY assumes that the cards following the DCK/ card are revision cards, and the COSY source deck follows the revision cards. COSY reads the revision cards and places them on the mass storage scratch area until an END/ card is read. Then COSY reads the new COSY source deck (which must follow the revision cards), and modifies the new deck according to the revision cards.

If I = lu is used and lu is not the system standard input, COSY reads the revision cards from the system standard input unit and the source deck specified by the DCK/ card from unit lu. Then COSY updates the source deck according to the revision cards.

C Parameter (COSY Output)

C = lu or C — C specifies the device to receive COSY output. If C is absent, there is no COSY output. If just C is used, COSY output is on the COSY standard output device. C cannot be equated to the unit containing the current COSY library.

H Parameter (Hollerith Output)

H = lu or H — H specifies the device receiving Hollerith output. If H is absent, there is no Hollerith output. If just H is used, Hollerith output is on the COSY standard output device.

D Parameter (Deck Name)

D = name — D changes the name of the COSY or Hollerith deck. COSY uses the six characters (including blanks and commas) following D = for the new deck name.

NOTE

If the name is less than six characters and an I, C, or H parameter follows it, COSY misinterprets the name.

id Parameter

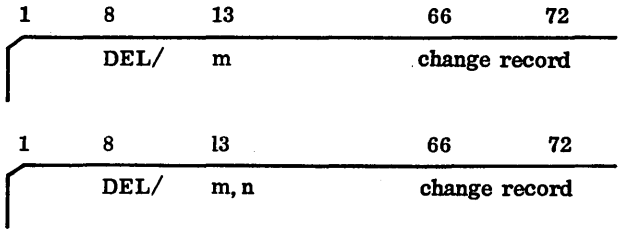
id — is the three-character field for changing the COSY or Hollerith deck identifier. If id is blank, the old deck identifier on the HOL/ or CSY/ card is used.

L Parameter (list)

L = lu or L — L specifies that a listing (in decompressed Hollerith form) of the deck be made on logical unit lu. If just L is used, the listing is on the COSY standard list device.

DEL/ Card

COSY deletes a specified number of cards from a previously defined input deck and inserts any Hollerith source cards immediately following the DEL/ card up to the next COSY control card. A DEL/ card has two forms:

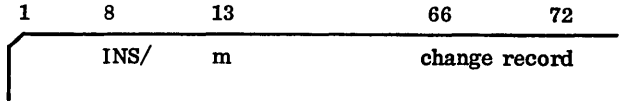


In the first form, card m is deleted; in the second, cards m through n are deleted. The unsigned decimal numbers m and n are the sequence numbers in columns 76 through 80 of the Hollerith source cards. Sequence number m must be less than n.

The number of Hollerith cards following a DEL/ card need not equal the number of cards being deleted.

INS/ Card

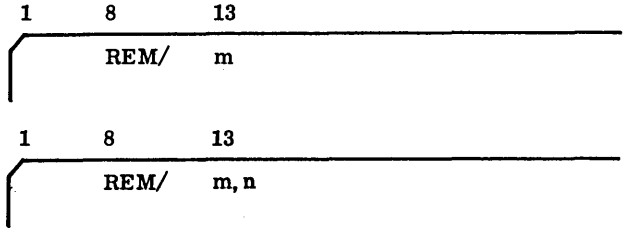
COSY inserts the Hollerith source cards immediately following an INS/ card into the new COSY or Hollerith deck.



The Hollerith source cards are inserted after sequence number M, which is found in columns 76 through 80 of the Hollerith source cards.

REM/ Card

The REM/ card is used to remove the DEL/ or INS/ card and all Hollerith source cards that follow. This operation occurs only when merging two revision decks. A REM/ card has two forms:

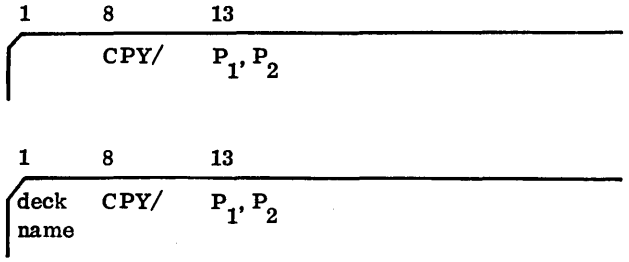


The sequence numbers m and n must match the sequence numbers on DEL/ or INS/ control cards in the revision deck that is being merged.

A REM/ card detected when COSY is not merging is ignored.

CPY/ Card

The CPY/ card causes the COSY library to be copied onto a logical output unit. The CPY/ card has two forms:



The first form, without the deck name, causes the COSY library to be copied from its current position to the end of the library. The second form, with a deck name specified, causes the COSY library to be copied from its current position through the named deck. CPY/ places and END/ card at the end of the new library, followed by an end-of-file mark.

The COSY library can be positioned to the beginning of any deck by the use of a CPY/ card on which only the deck name and the I parameter are specified. This card positions the COSY library to the beginning of the deck, immediately following the named deck.

The p parameters specify the logical I/O units used to copy the COSY library. These parameters can occur in any order and are in the form p = lu, where:

p = I or C
lu = a logical I/O unit

I = lu or I	I specifies the logical unit, lu, from which the COSY library is copied. If the I parameter is omitted or just I is used, the COSY library is copied from the COSY standard input device.
C = lu or C	C specifies the logical unit, lu, to which the COSY library is copied. If just C is used, the COSY library is copied onto the COSY standard output device. If C is omitted, there is no COSY output.

As each COSY deck is read from input unit I and copied on output unit C, the deck name is listed on the COSY standard print device. For example:

```
Deck name        CSY/    *COPIED*
```

For each deck that is read but not copied, the *COPIED* notation is omitted. For example:

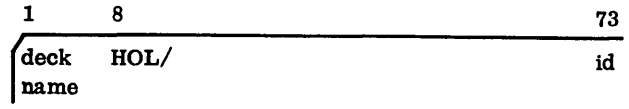
```
Deck 1      CSY/      *COPIED*
Deck 2      CSY/      *COPIED*
.           .         .
.           .         .
.           .         .
Deck 9      CSY/      *COPIED*
Deck 10     CSY/                  }  Decks 10
.           .                     }  through 14
.           .                     }  were read but
.           .                     }  not copied.
Deck 14     CSY/                  }
Deck 15     CSY/      *COPIED*
Deck 16     CSY/      *COPIED*
.           .         .
.           .         .
.           .         .
```

END/ Card

The END/ card terminates Hollerith input decks, COSY libraries, Hollerith input libraries, and revision decks.

HOL/ Card

When a Hollerith deck is input, the first card must be a Hollerith deck identifier.

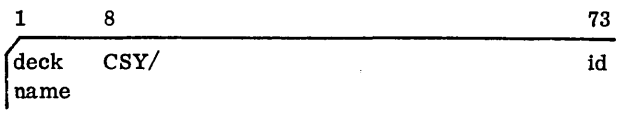


deck name	Names the COSY deck being processed
id	Three-character deck identifier

A Hollerith deck identifier is not produced for a Hollerith output deck.

CSY/ Card

When COSY output is requested on the DCK/ card, COSY generates a COSY deck identifier card as the first card of the COSY output deck. COSY deck identifiers must also precede COSY DECKS on input.



deck name	Names the COSY deck being processed
id	Three-character deck identifier or original deck

SAMPLE COSY REVISION DECKS

The following sample COSY revision decks illustrate the use of COSY control cards.

Generating A Cosy Library

Figure 14-7 generates a COSY library from two Hollerith source decks and places the library on output unit 3. The system standard input unit (card reader) is unit 1.

Updating Cosy Desks

Figure 14-8 updates three COSY decks and places the updated decks on logical unit 7. Two of the COSY decks are on logical unit 6 and the third deck (deck 3) is input following the revision decks. The system standard input unit (card reader) is unit 1.

Using the CPY/ Card to Update COSY Library

Figure 14-9 is an example of updating a COSY library by using the CPY/ card to replace five old COSY decks with five new COSY decks. Logical unit 5 contains the old COSY library (decks 1 through 24), and logical unit 4 contains five replacement decks. The new COSY library is output on logical unit 6.

Merging Two Revision Decks

Figures 14-10 and 14-11 contain two examples of merging revision decks. Example 1 (figure 14-10) merges two decks that appear on the same input unit. Example 2 (figure 14-11) merges two decks that appear on different input units.

Example 1

This job merges two revision decks (A and B), which appear on logical unit 2 (card reader), and writes the merged output as a revision deck in Hollerith format on logical unit 3 (see figure 14-10).

Example 2

This job merges two revision decks (A and B) and writes the merged revision deck on logical unit 3 (magnetic tape). Revision deck A is on logical unit 5 (card reader) and revision deck B is on logical unit 2 (punched paper tape). See figure 14-11.

Since deck A was the primary deck, the DEL/ 10, 11 card and the insert cards following it take precedence over the DEL/ 10, 11 card and insert cards in deck B. The REM/ 21, 22 card in deck A removes the DEL/ 21, 22 card and the following source cards from deck B. The DEL/ 29 card from deck A is added to the merged revision deck.

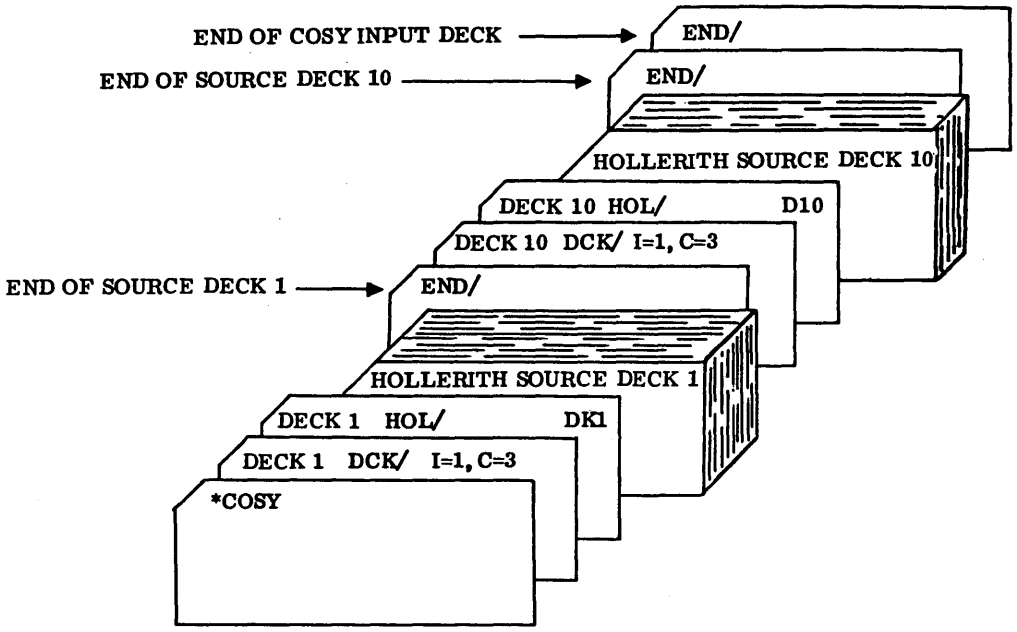


Figure 14-7. Example of Generating a COSY Library

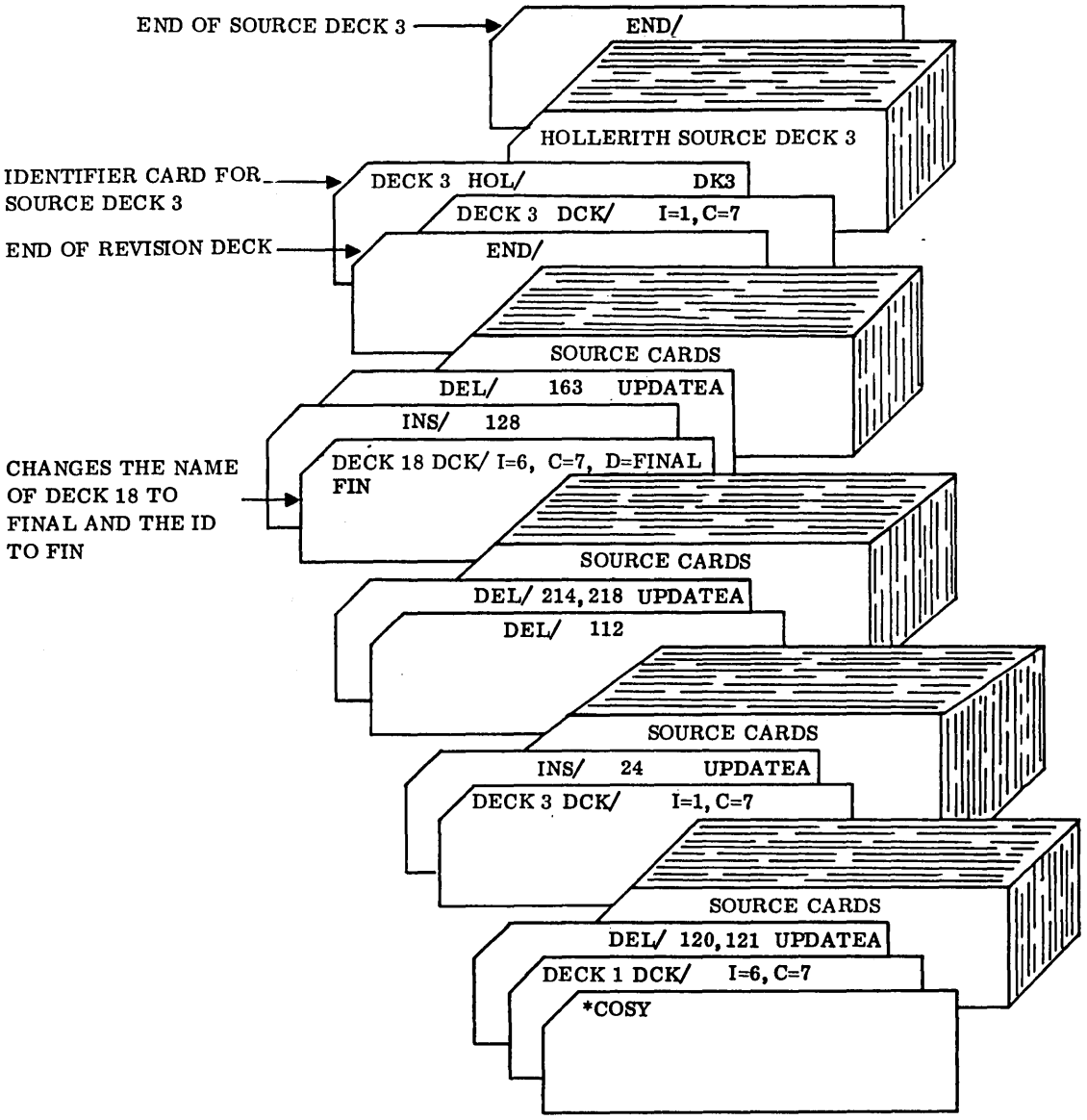


Figure 14-8. Example of Updating COSY Decks

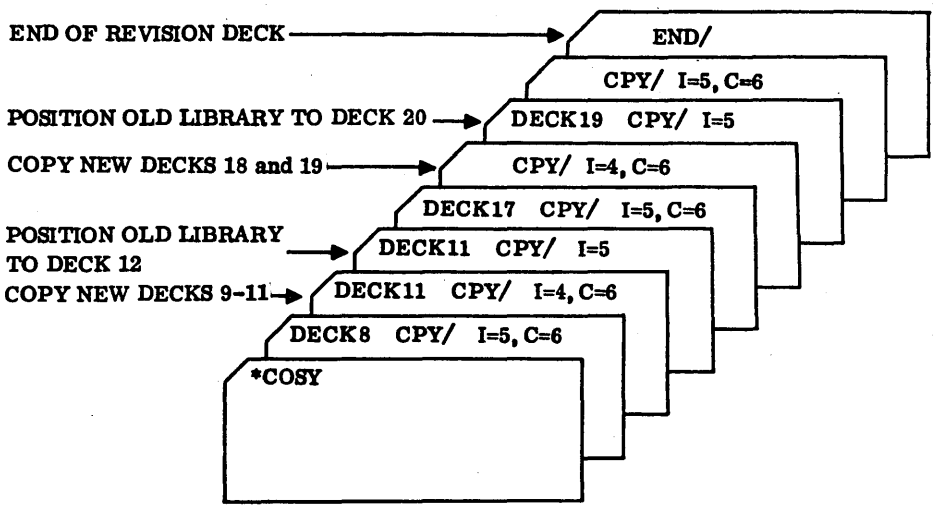


Figure 14-9. Example of Using the CPY/ Card to Update a COSY Library

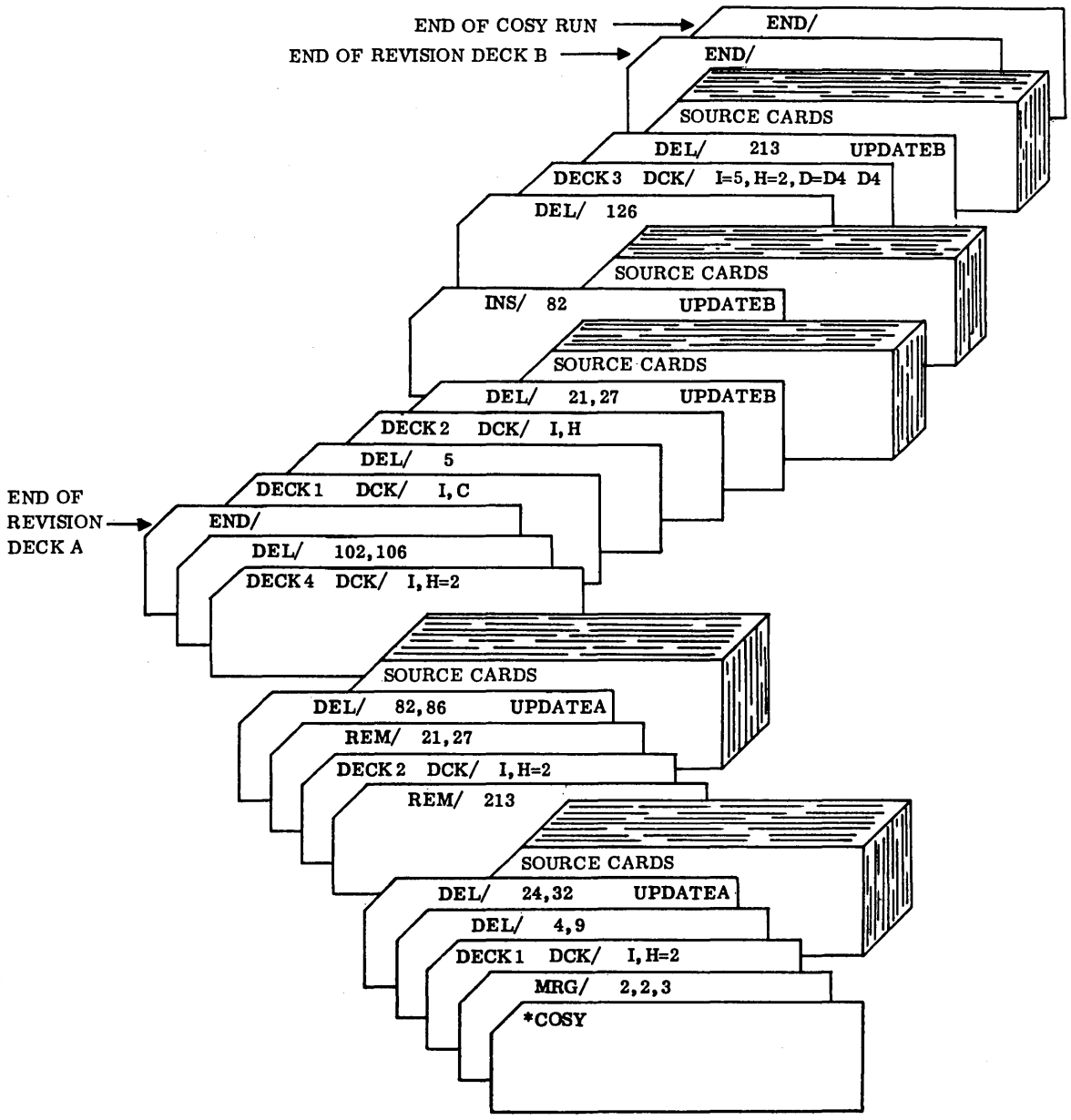


Figure 14-10. Example of Merging Two Revision Decks from Same Input Unit

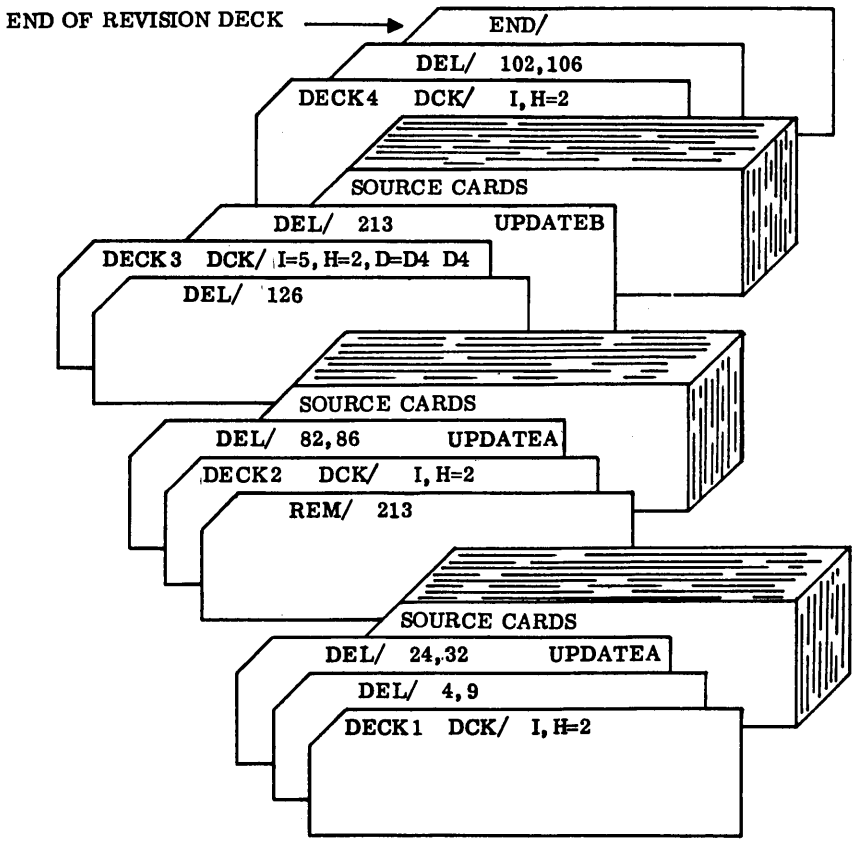
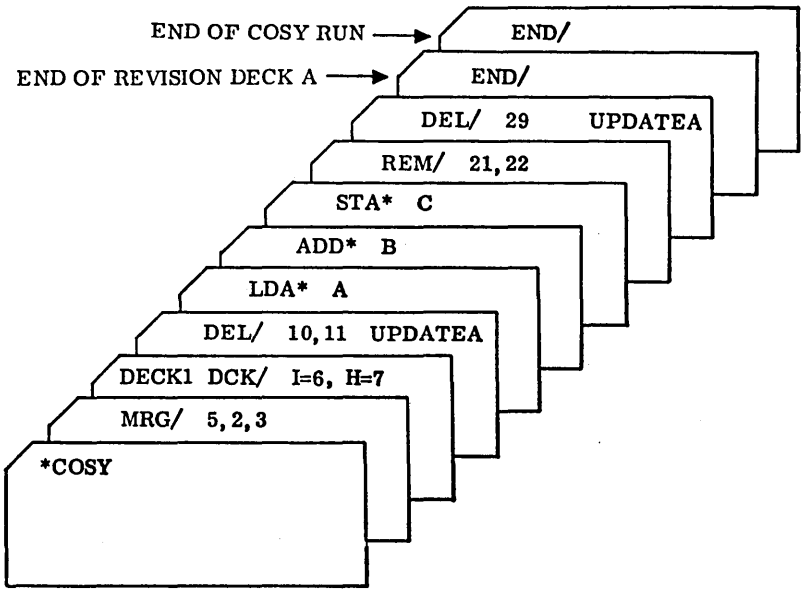


Figure 14-10. Example of Merging Two Revision Decks from Same Input Unit (Contd)



REVISION DECK A (INPUT FROM CARD READER)

```
DECK1       DCK/    I=6,H=7
            DEL/    10,11       UPDATEB
            LDA*    A
            SUB*    B
            STA*    C
            DEL/    21,22       UPDATEB
```

(SOURCE CARDS FOR INSERTION BETWEEN COSY CARDS 20 AND 23)

```
            END/                (END OF REVISION DECK B)
```

REVISION DECK B (INPUT FROM PAPER TAPE)

```
DECK1       DCK/    1=6, H=7
            DEL/    10,11       UPDATEA
            LDA*    A
            ADD*    B
            STA*    C
            DEL/    29          UPDATEB
            END/                (END OF MERGED
                                REVISION DECK)
```

MERGED REVISION DECK (OUTPUT ON MAGNETIC TAPE)

Figure 14-11. Example of Merging Two Revision Decks from Different Input Units

Converting COSY Decks to a Hollerith Library

Figure 14-12 is an example of a job that converts three COSY decks into a Hollerith library and writes the Hollerith library on logical unit 6. COSY decks 1 and 2 are on logical unit 5 and COSY deck 3 is on logical unit 7.

COSY LIBRARY

The COSY library consists of one or more COSY decks and is terminated with an END/ card. The COSY deck is a series of compressed source written in ASCII format. Each COSY deck begins with a COSY deck identifier and ends with an end-of-deck character followed by and end-of-file mark.

COSY compresses a card image by inserting a special ASCII character and value for three or more sequential blanks.

$5F_{16}$	Special ASCII character indicating compression
$5Fxx_{16}$	Indicates 3 to 62 consecutive blanks, where $21_{16} \leq xx \leq 5D_{16}$ (except 26_{16}, which is illegal)
$5F5E_{16}$	End-of-card image
$5F5F_{16}$	End- of-deck image

The format allows COSY to process all ASCII characters on paper tape that are valid for the device. Illegal paper tape characters are 00, 7F, 09, 0A, 0B, 0C, and 0D. However, the only valid ASCII characters on magnetic tape are $20_{16} \leq xx \leq 5F_{16}$ (except 26_{16}). This set includes all capital letters and all numbers. ASCII characters 00_{16} through $1F_{16}$ and 60_{16} through $7F_{16}$ are illegal because they cannot be written on magnetic tape in BCD mode. The character 26_{16} is illegal because it produces blank tape in BCD mode.

The COSY library may be written on paper tape or magnetic tape. When the library is on paper or magnetic tape, the block size is 192 words and all blocks are completely filled. Source cards images may be split across blocks.

HOLLERITH INPUT

A Hollerith input library is a group of (one or more) Hollerith source decks which is terminated by an END/ card. Each Hollerith source deck begins with a Hollerith deck identifier card and ends with an END/ card. The Hollerith input may be input from cards, paper tape, magnetic tape, or teletypewriter.

HOLLERITH OUTPUT

Hollerith output consists of source decks in uncompressed Hollerith code produced from COSY decks. Columns 73 through 75 of the source cards contain a deck identifier. Asterisks appear in this field if the source card was inserted by a revision deck.

Columns 76 through 80 of the Hollerith source cards contain a decimal sequence number. If new source cards are inserted with a revision deck containing DEL/ or INS/ cards, COSY writes an asterisk in column 73 of each new source card and writes the change record field (contents of columns

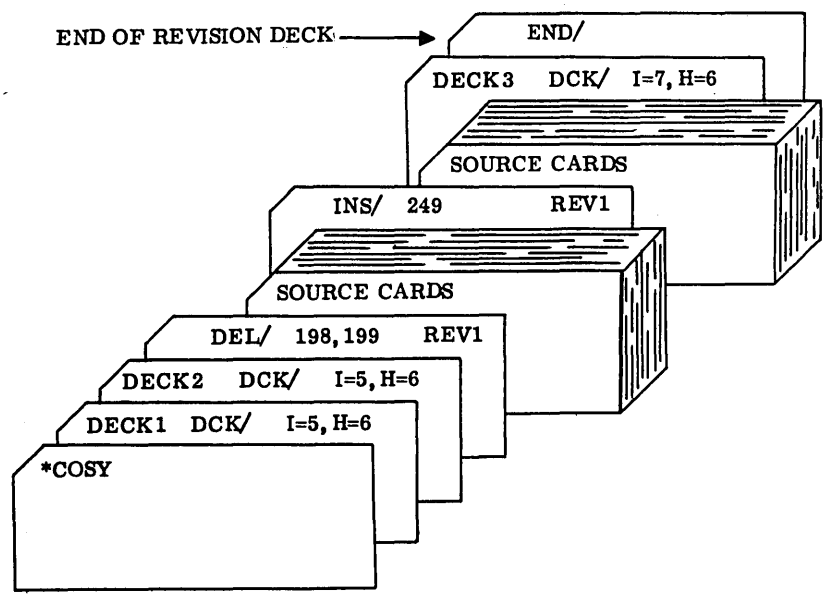


Figure 14-12. Example of Converting COSY Decks to a Hollerith Library

66 through 72 on the DEL/ or INS/ card) in columns 74 through 80 of the new source cards. If the change record field was blank on the INS/ or DEL/ cards, COSY fills columns 73 through 80 with asterisks.

Hollerith output may be on punched cards, punched paper tape, or magnetic tape, and is terminated with an end-of-file mark. If the COSY output is on magnetic tape, COSY writes an end-of-file mark and rewinds the tape upon completion of the COSY run.

REVISION DECK

A revision deck is a group of COSY control cards and new source cards that are used to update or revise an existing COSY library. The first card of a revision deck must be a DCK/, MRG/, or CPY/ control card and the last card must be an END/ control card. The new source cards, if used, must follow an INS/ or DEL/ control card. The control cards are described in the Calling Statements section. All cards are in Hollerith code.

The revision deck is input to COSY on the system standard input device. If the source deck to be revised is on the system standard input device, COSY stores the revision deck or mass storage scratch until the source deck has been read. The revision deck is stored as card images with 40 words per sector.

LISTINGS

Under normal operation, COSY lists revisions from the revision deck as they occur on input. However, when merging two revision decks, COSY lists the final merged revision deck on the standard print device. Asterisks in columns 2 through 5 indicate the card was not used in the COSY operation. Columns 6 through 85 contain the revision input card.

If the L parameter is not present on the DCK/ card and revision cards follow the DCK/ card, the revision cards are listed.

MESSAGES

COSY error messages are written on the COSY standard list device in the following format:

```
COSY Cxx
```

Where: xx is the error code.

Refer to the MSOS Diagnostic Handbook for descriptions of the COSY error messages.

At the end of a COSY job, the following message is written on the COSY standard list device (only if errors exist):

```
xx ERRORS.
```

Where: xx is a decimal count of errors in the COSY job.

At various times during a COSY job, the following message may be written on the system standard comment device:

```
REWIND LU xx
```

Where: xx (decimal) indicates the logical unit to be rewound.

The operator may enter any value through the system standard input comment device after rewinding the unit to tell COSY that the unit has been rewound.

LIST COSY PROGRAM (LCOSY)

The list COSY program provides a means of listing the names of programs on a COSY tape and punching DCK/ control cards for each program.

The procedure to execute this program is:

```
*JOB
J
*K,Ilu,Plu,Llu
J
*LCOSY
```

Where: I assigns a logical unit of COSY tape to be read.

P assigns the logical unit where DCK/ control statements are to be punched; the END/ statement is not punched.

L assigns the logical unit where names of programs are listed. The listing appears in the following format:

```
PBY         CSY/
PBYA        CSY/
PBZ         CSY/
PBZA        CSY/
END OF COSY LIBRARY
```

Upon executing LCOSY, the following typeout occurs:

```
DCK/  I,H,C
```

LCOSY waits for a two-digit logical unit number for each of the parameters separated by a comma; for example: 06, 07, 18.

If a parameter is not desired, a slash replaces the logical unit number; for example: 06,/,18. If a slash is used under the I parameter, no DCK/ control cards are punched; only a listing of the deck names from the input source is produced.

DISK-TO-TAPE LOADER (DTLP)

The disk-to-tape loading program provides the means of loading the disk-to-tape (DSKTAP) program into core. DSKTAP must be absolutized and put on the disk prior to using DTLP.

The procedure to execute this program is:

```
*JOB
J
*DTLP
```

DSKTAP is put into operation by a jump to the location at which DSKTAP is loaded into core.

DISK-TO-TAPE UTILITY (DSKTAP)

The disk-to-tape program is a stand-alone, off-line program. When executing, the protect switch must be off because protected core is used as a buffer area. The program has its own drivers for the following equipment:

- Disk – One of the following device drivers is loaded:

  -1833-1/1833-3/1867-10/1867-20 Disk
  -1833-4/1866-12/1866-14 Cartridge Disk
  -1733-1/1738 Disk
  -1733-2 Disk
  -1739-1 Disk
  -1752 Drum

- Tape – One of the following device drivers is loaded.

  -1860-95 Magnetic Tape
  -1860-92 Magnetic Tape
  -1860-72 Magnetic Tape
  -All other seven- and nine-track tapes

When DTLP is called from the program library for execution, it does a GTFILE request for the file DSKTAP and prints the following message on the standard comment device.

DTLP FIRST WORD ADDRESS WILL BE xxxx

Where: xxxx is the core location.

DTLP then prints the followimg message on the comment device and gives control to DSKTAP:

TURN OFF PROTEC SWITCH, TYPE CARRIAGE RETURN

DSLTAP then enters into the following dialogue with the operator:

4 DIG. EQ. CODE FOR.. (1)
MAG TAPE

Response to this message is the four-digit hexadecimal equipment code. For example:

0381 for 1700 systems

0480 for CYBER 18 systems

For a buffered tape, it might be:

1381

The following message is then entered:

4 DIG. EQ. CODE FOR.. (2)
MASS MEMORY

Respond by typing the four-digit hexadecimal equipment code for the mass-memory controller. For example, an equipment code of 3 for 1700 or 14 for CYBER 18 requires the following to be entered:

1811 for 1700 Systems

0700 for CYBER 18 Systems

If either equipment code contains a nonhexadecimal digit, the following message is printed:

ILLEGAL PARAMETER SPECIFIED (3)

Control then reverts to message 1.

When the equipment codes have been properly specified, the message:

SCRATCH SECTOR IN $C1 IS xxxx (4)

is typed with xxxx being the current beginning of scratch mass memory. If a system is being saved on tape, all mass memory sectors up to xxxx should be saved.

The next message is self-explanatory.

TYPE LOAD FOR TAPE-TO-DISK, SAVE FOR DISK-TO-TAPE OR CARRIAGE RETURN (5)

If LOAD is the response, control goes to message 6. If SAVE is the response, control goes to message 7. If a carriage return is the response, control goes to message 11.

INPUT TAPE ON UNIT 0. READY/ (6)

Response with a carriage return when ready.

OUTPUT TAPE ON UNIT 0. (7)
HOW MANY SECTORS?

Response with the four-digit hexadecimal number of sectors to be saved.

If the mass memory device does not respond properly to I/O commands, the following message is typed and control passes to message 5.

```
DISK ERROR (xxxx)                    (8)
```

Where: xxxx is the last device status.

If the magnetic tape does not respond, the following message is printed and control passes to message 5.

```
TAPE ERROR (xxxx)                    (9)
```

Where: xxxx is the last device status.

When the LOAD operation has completed, the following message is printed and control passes to message 11.

```
xxxx SECTOR LOADED                   (10)
```

When the SAVE operation is completed, the following message is printed:

```
TYPE V FOR VERIFY,                   (11)
A FOR AUTOLOAD, OR
A CARRIAGE RETURN
```

If the response to message 11 is an A, the program simulates an autoload by reading the first track of the mass-memory device to location 0 and jumping to zero. This is the only exit from DSKTAP.

If the response is a V, the tape is verified against mass memory when the following message is responded to with a carriage return.

```
VERIFY TAPE ON UNIT 0. READY?        (12)
```

When the verification is complete, control goes to message 11. When errors are encountered on verify, control goes to message 13.

Only the first verify error in a block of 16 sectors is logged. Verify errors cause the following message:

```
SECTOR xxxx WORD — ww — DOES         (13)
NOT COMPARE. TYPE C TO CON-
TINUE OR A CARRIAGE RETURN
TO ABORT.
```

Where: xx is the sector and ww is the word within the sector.

When a carriage return is the response, control passes to message 11.

NOTE

On the CYBER 18, the DTLP program must be loaded into the lower 32K of core. If the background area is in the upper bank area, the DTLP program moves itself to the lower 32K before execution.

INPUT/OUTPUT UTILITY (IOUP)

The CYBER 18/1700 background I/O utility program (IOUP) enables the user to perform simultaneous peripheral background operations during normal foreground processing via requests entered at the standard input or comment device. Since the IOUP program operates under CYBER 18/1700 MSOS, the operating system's drivers must be available for all I/O equipment used in the IOUP program.

IOUP STATEMENT

The IOUP program runs in the background and is called by the job processor when the operator types an *IOUP statement. IOUP reads control statements from the standard input device. The operator may switch control to the standard comment device by entering an *K,14; it continues processing until terminated by either an OUT or an *Z statement, followed by a RETURN.

All operator requests are limited to 40 characters on the input device.

An *IOUP statement initiates the program that alerts the operator that it is in core and ready for use by typing out:

```
UTILITY IN
NEXT IOU
```

After the request is completed and all messages associated with the completed requests have been typed out, the IOUP program types out:

```
NEXT IOU
```

The IOUP program then waits for the next request form the operator. To terminate the IOUP program, the operator must:

```
Type     OUT
Press    RETURN
```

The IOUP program types out:

UTILITY OUT

To abort a request in execution, the operator must

Press	MANUAL INTERRUPT (on console)
Type	*Z
Press	RETURN

The job processor terminates the processing of the request.

THEORY OF OPERATION

An IOUP request performs three tasks:

- Transfers data
- Compares data
- Requests motion control

The transfer and comparison of data is performed as follows: data is read form one logical unit and written to or compared with the data on the second logical unit. Multiple copies of the input data can be made on the output device.

The tape motion request may be: skip the specified number of files/records forwards/backwards, set the density, write and end-of-file, rewind, or unload a magnetic tape.

The IOUP program executes these tasks on the following types of devices:

- Card reader
- Card punch
- Magnetic tape
- Paper tape reader
- Paper tape punch
- Line printer

References to the I/O devices used by the IOUP are made by logical units or the standard I, P, or L notation assigned by the system or the job processor.

Data Transfer

The IOUP program employs two methods for the transfer of data. The method to be used is governed by the following:

- Intermediate storage is used if the input and output units are the same and/or multiple copies of the input data are requested (method 1).
- Intermediate storage is not used (record-by-record transfer occurs) if the input and output units are not the same and only one copy of the input data is requested (method 2).

Method 1

Read the entire input data from the input and write to intermediate storage (scratch unit). For transfer requests the input data from the intermediate storage is read one record at a time and then output to the specified device. If multiple copies of the output are required, the intermediate storage is read and output until the repeat count is satisfied.

Using this method, the amount of data to be transferred is restricted by the size of the available scratch unit. If the input data to be transferred is specified to be larger than the available scratch, or if it is in the process of writing, the scratch is exhausted and the request is aborted.

Method 2

The intermediate storage is not used in the execution of an IOUP request.

Transfer of data from one record of the input device is read and written directly to the specified output device. This record-by-record transfer continues until either the specified amount of data is transferred or a physical end of the input device is detected. Only one copy of the data can be made.

Data Verification

The IOUP program ses one of the following methods for verification of data:

- Comparison of data with two of the same logical units.
- Comparison of data with two different logical units.

Method 1

When the two logical units are the same, the entire data from the first logical unit is read and written on intermediate storage. This continues until all of the specified number of records are read and written or until the end of data on either logical unit is detected. After completion of reading from the first logical unit and writing on intermediate storage, one record at a time is read from the second logical unit and compared with the imtermediate storage for record length and data matching. This comparison continues until the end of the data on the second unit or to the end of intermediate storage.

When a mismatch occurs, a diagnostic is output (refer to the MSOS Diagnostic Handbook).

Method 2

When the data to be compared is on two different logical units, a record from the first logical unit is read and compared with the corresponding record of the data on the second logical unit. The two records are checked for the same number of words and word-by-word likeness. The comparison continues until all of the specified number of records are compared or until the end of data on either logical unit is detected.

When a mismatch occurs, a diagnostic is output (refer to the MSOS Diagnostic Handbook).

Data Record Size

The maximum data record size for an IOUP request is 192 words (assembly parameter). The I/O devices used by this request are:

- Card reader/punch
- Paper tape reader/punch
- Magnetic tape

PERIPHERAL OPERATIONS PERFORMED BY THE IOUP PROGRAM

The following is a summary of the peripheral operations performed by the IOUP program. They are discussed in detail in the remainder of the section.

Data Transfer Requests:

Request	Call Format
Card to card	CC,u_1,u_2,m,x
Card to magnetic tape	CM,u_1,u_2,m,x
Card to paper tape	CP,u_1,u_2,m
Card to printer	CL,u_1,u_2,m,x
Paper tape to printer	PL,u_1,u_2,A/B,n,m
Paper tape to paper tape	PP,u_1,u_2,n,m
Paper tape to magnetic tape	PM,u_1,u_2,A/B,n,m
Paper tape to card and printer	PB,u_1,u_2,u_3,A/B,n,m
Paper tape to card	PC,u_1,u_2,A/B,n,m
Magnetic tape to card	MC,u_1,u_2,R/F,n,m,x
Magnetic tape to printer	ML,u_1,u_2,R/F,n,m,x
Magnetic tape to card and printer	MB,u_1,u_2,u_3,R/F,n,m,x
Magnetic tape to magnetic tape	MM,u_1,u_2,R/F,n,m
Magnetic tape to paper tape	MP,u_1,u_2,R/F,n,m

Data Verification Requests:

Request	Call Format
Card and card	VCC,u_1,u_2,x
Card and paper tape	VCP,u_1,u_2
Card and magnetic tape	VCM,u_1,u_2,n,x
Paper tape and paper tape	VPP,u_1,u_2
Magnetic tape and magnetic tape	VMM,u_1,u_2,n
Magnetic tape and paper tape	VMP,u_1,u_2,n

Motion Control Requests:

Request	Call Format
Advance unit number of files	TAF,u,n
Advance unit number or records	TAR,u,n
Backspace unit number of files	TBF,u,n
Backspace unit number of records	TBR,u,n
Rewind unit	TRW,u
Write end-of-file mark on unit	TEF,u

Set density of unit	TSD,u,d
Unload unit	TUL,u

The u, u_1, u_2, and u_3 of all the preceding IOUP requests indicate the devices used by the job processor to execute an IOUP request. If the type of the device indicated by I, P, L, u, u_1, u_2, or u_3 does not match the type of IOUP request, the request is rejected.

Data Transfer Requests

Card to Card

The card-to-card data transfer request format is:

CC,u_1,u_2,m,x

Where: u_1 is card reader unit 1.
u_2 is punch unit 2.
} u_1 and u_2 can refer to the same unit.

m is the number of times the input deck is to be copied.

x is optional.

0 or blank	Format of input data is 1700 formatted binary/ASCII.
1 or 99999	80-column card image in binary

Card deck input at card reader unit u_1 is duplicated m times at the card punch unit u_2.

The end of input data occurs when an end-of-file is detected or the card reader hopper is empty. The number of records (one record = one card) transferred is typed out after the completion request.

Card to Magnetic Tape

The card-to-magnetic-tape data transfer request format is:

CM,u_1,u_2,m,x

Where: u_1 is the card reader unit.

u_2 is the magnetic tape unit.

m is the number of times the input deck is to be copied.

x is optional.

0 or blank	Format of input data is 1700 formatted binary/ASCII.
1 to 99999	80-column card image in binary

Card deck input at card reader unit u_1 is written m (1 to 10) times on the magnetic tape on u_2.

An end-of-file mark is written at the end of each copy of the magnetic tape unit.

The end-of-input information occurs when an end-of-file card is detected or the card reader hopper is empty.

When an end-of-tape condition on magnetic tape is detected, a diagnostic is printed. At the end of each copy the following message is output:

COPY COMPLETED, (cr) WHEN READY FOR NEXT COPY.

Where: (cr) is carriage return.

Data is written on the u_2 magnetic tape, beginning at the current physical position of the tape. The tape density is determined by the setting on unit u_2. The number of records (one record = one card) transferred is typed on completion of the request.

Card to Paper Tape

The card-to-paper-tape data transfer request is:

CP,u_1,u_2,m

Where: u_1 is the card reader unit.

u_2 is the paper tape punch.

m is the number of copies of paper tape required.

Card deck input at card reader unit u_1 is written m times on the paper tape on unit u_2. The mode of the data transfer is the mode of the input data; 1700 binary/ASCII mode is assumed.

A leader is written at the end of each copy on the paper tape. The end of the input information occurs when an end-of-file card is detected or the card reader is empty.

The number of records (one record = one card) of data transferred is typed on completion of the request.

Card to Printer

The card-to-printer data transfer request format is:

CL,u_1,u_2,m,x

Where: u_1 is the card reader unit.

u_2 is the printer.

m is the number of listings required.

x is optional.

0 or blank	The first character of the card record is not interpreted as a carriage control function for a FORTRAN line printer; it is printed as a data character.
1 to 99999	The first character of the card record is interpreted as a carriage control function for a FORTRAN line printer.

Card deck input at card reader unit u_1 is written m times on the printer of unit u_2.

The mode of the data transfer is assumed to be 1700 formatted binary/ASCII mode used for the input card deck. Non-ASCII input data results in a garbled printout.

A paper eject occurs at the end of each listing.

The end of the information occurs when an end-of-file card is detected or the card reader hopper is empty.

The number of records (one record = one card) is typed on completion of the request.

Paper Tape to Printer

The paper-tape-to-printer data transfer request format is:

PL,u_1,u_2,A/B,n,m

Where: u_1 is the paper tape reader.

u_2 is the printer.

A/B is the mode of the data on paper tape.

A	ASCII
B	1700 binary

n is the number of records of data to be printed.

m is the number of listings required.

The mode of the data transfer is that specified in the request and is assumed to be 1700 formatted binary/ASCII. The paper tape on unit u_1 is read in and n (1 to 99,999) records of data (or the actual number read if there are records less than the specified number) are printed on printer unit u_2.

After each listing a paper eject occurs.

The paper tape is read in the specified mode by a format read request.

The number of records transferred is typed out on completion.

The printout of binary input data is garbled.

Paper Tape to Paper Tape

The paper-tape-to-paper-tape data transfer request format is:

PP,u_1,u_2,n,m

Where: u_1 is the paper tape reader.

u_2 is the paper tape punch.

n is the number of records of data to be transferred (1 to 99,999).

m is the number of times the paper tape is to be duplicated.

The number of records of data on paper tape unit u_1 is read in and duplicated m times on paper tape punch unit u_2.

Paper Tape to Magnetic Tape

The paper-tape-to-magnetic-tape data transfer request format is:

PM,u_1,u_2,A/B,n,m

Where: u_1 is the paper tape reader.

u_2 is the magnetic tape unit.

A/B is the mode of data on paper tape.

A	ASCII
B	1700 binary

n is the number of records to be transferred.

m is the number of times the paper tape is to be duplicated.

At the end of each copy the following message is output:

COPY COMPLETED, (cr) WHEN READY FOR NEXT COPY.

Where: (cr) is carriage return.

The paper tape is read by a format read request in the mode specified in the request. The density of recording data on magnetic tape is the setting on the physical magnetic tape unit.

The paper tape unit u_1 is read in and n (1 to 99,999) or less (if the number of records actually read is less than n) records of data are written on magnetic tape unit u_2 in the specified mode; 1700 formatted binary/ASCII is assumed.

An end-of-file mark is recorded after each copy of the data on the magnetic tape. When an end-of-tape mark is detected on the magnetic tape, a diagnostic is printed.

The number of records transferred is typed on completion of the request.

Paper Tape to Card and Printer

The paper-tape-to-card-and-printer data transfer request format is:

PB,u_1,u_2,u_3,A/B,n,m

Where: u_1 is the paper tape reader.

u_2 is the card punch unit.

u_3 is the printer.

A/B is the mode of data on paper tape.

A	ASCII
B	1700 binary

The mode of the data transfer is that specified in the request. The paper tape is read by a format read in the specified mode; 1700 formatted binary/ASCII is assumed.

n is the number of records of data to be transferred.

m is the number of listings and punched card decks required.

The paper tape or unit u_1 is read in and n (1 to 99,999) or less (if the number actually read is less than the specified n value) records of data are printed on printer unit u_3 and punched on card punch unit u_2.

After each listing a paper eject occurs.

The number of records transferred is typed on completion of the request. The printout of binary input data is garbled.

Paper Tape to Card

The paper-tape-to-card data transfer request format is:

PC,u_1,u_2,A/B,n,m

Where: u_1 is the paper tape reader.

u_2 is the card punch unit.

A/B is the mode of data on paper tape.

A	ASCII
B	1700 binary

n is the number of records of data to be transferred.

m is the number of punched card decks to be duplicated.

The paper tape on unit u_1 is read in and n (1 to 99999) or, less (if the number actually read is less than the specified n value) records of data are punched on card punch unit u_2.

The paper tape is read as formatted records in the specified mode and the cards are punched as formatted records in the mode of the input data; 1700 formatted binary/ASCII format is assumed.

The number of records transferred is typed on completion of the request.

Magnetic Tape to Card

The magnetic-tape-to-card data transfer request format is:

MC,u_1,u_2,R/F,n,m,x

Where: u_1 is the magnetic tape unit.

u_2 is the card punch unit.

R is the number of records (n) to be transferred (1 to 99,999).

F is the number of files (n) to be transferred (1 to 99,999).

n is the number of records/files to be transferred (refer to R, F)

m is the number of card decks to be produced.

x is optional.

0 or blank	Format of input data is 1700 formatted binary/ASCII.
1 to 99999	Binary card image (160 tape characters/record) is punched as an 80-column card record.

The magnetic tape on unit u_1 is read in and the specified number of records/files are punched on card punch unit u_2.

The number of records/files transferred is less than the specified number if the end-of-tape is detected before all specified input data has been read in.

The number of records/files transferred is typed on completion of the request.

Magnetic Tape to Printer

The magnetic-tape-to-printer data transfer format is:

ML,u_1,u_2,R/F,n,m,x

Where: u_1 is the magnetic tape unit.

u_2 is the printer.

R is the number of records (n) to be transferred.

F is the number of files (n) to be transferred.

n is the number of records/files to be transferred (refer to R, F).

m is the number of listings required.

x is optional.

0 or blank	First character of magnetic tape record is not interpreted as a carriage control function. It is printed as data character.
1 to 99,999	The first character of magnetic tape record is interpreted as a carriage control function for a FORTRAN line printer.

The magnetic tape on unit u_1 is read in and the specified number of records/files is printed on printer u_2.

Each end-of-file encountered on magnetic tape causes a page eject.

At the end of each listing a page eject occurs.

The number of records/files transferred is less than the specified number if the end-of-tape is detected before all specified input data has been read in.

The mode of data transfer is 1700 formatted binary/ASCII mode of data on magnetic tape. The magnetic tape is read by a format read request and the data is written in the mode of the input data. The printout of binary input data is garbled.

The number of records/files transferred is typed on completion of the request.

Magnetic Tape to Card and Printer

The magnetic-tape-to-card-and-printer data transfer request format is:

MB,u_1,u_2,u_3,R/F,n,m,x

Where: u_1 is the magnetic tape unit.

u_2 is the card punch unit.

u_3 is the printer.

R is the number of records (n) to be transferred.

F is the number of files (n) to be transferred.

n is the number of records/files to be transferred (1 to 99,999).

m is the number of listings and card decks required.

x is optional.

0 or blank	First character of the magnetic tape record is not interpreted as a carriage control function for a FORTRAN line printer. It is printed as a data character.
1 to 99,999	First character of the magnetic tape record is interpreted as a carriage control function for a FORTRAN line printer.

The magnetic tape on unit u_1 is read in and the specified number of records/files are output on printer u_3 and punched on card punch unit u_2.

Each end-of-file encountered on the magnetic tape causes a page eject on the printer. At the end of each listing a page eject occurs.

The number of records/files transferred is less than the specified number n if the end-of-tape is detected before all specified data is read in.

The mode of data transfer is 1700 formatted binary/ASCII mode on magnetic tape. The magnetic tape is read by a format read request and the data is output to the printer and card punch units in the mode of the input data. The printout of binary input data is garbled.

The number of records/files transferred is typed on completion of the request.

Magnetic Tape to Magnetic Tape

The magnetic-tape-to-magnetic-tape data transfer request format is:

MM,u_1,u_2,R/F,n,m

Where: u_1 is the input magnetic tape unit.

u_2 is the output magnetic tape unit.

} u_1 and u_2 can refer to the same unit.

R is the number of records (n) to be transferred (1 to 99,999).

F is the number of files (n) to be transferred (1 to 99,999).

n is the number of records/files to be transferred (refer to R, F).

m is the number of copies to be made.

The magnetic tape in unit u_1 is read in and the specified number n of records/files is written m times on magnetic tape unit u_2. u_1 and u_2 can refer to the same unit.

The mode of data transfer is that of the data on the input magnetic tape. The magnetic tape is format read in 1700 binary/ASCII mode and the data is written in the format mode of input data. At the end of each copy of data, an end-of-file mark is written on the output magnetic tape. Each end-of-file mark detected on the input magnetic tape is counted as one record when R is specified. The number of records/files transferred is less than the specified number if the end-of-tape mark is sensed on the input magnetic tape before n is satisfied.

When an end-of-tape mark is detected on the output magnetic tape u_2, a diagnostic is typed.

The total number of records/files transferred is printed on completion of the request. At the end of each copy the following message is output:

COPY COMPLETED, (cr) WHEN READY FOR NEXT COPY.

Where: (cr) is carriage return.

Magnetic Tape to Paper Tape

The magnetic-tape-to-paper-tape data transfer request format is:

MP,u_1,u_2,R/F,n,m

Where: u_1 is the magnetic tape unit.
u_2 is the paper tape punch unit.
(u_1 and u_2 can refer to the same unit.)

R is the number of records (n) to be transferred (1 to 99,999).

F is the number of files (n) to be transferred (1 to 99,999).

n is the number of records/files to be transferred (refer to R, F).

m is the number of times the paper tape is to be duplicated.

The magnetic tape on unit u_1 is read in and the specified number n of records/files is written n times on the paper tape on unit u_2.

The mode of data transfer is that of the data on the input magnetic tape. The magnetic tape is format read in 1700 binary/ASCII mode and the data is format-punched as 1700 binary/ASCII. At the end of each copy of data, a leader of blank characters is punched. An end-of-file mark on the magnetic tape causes a leader to be punched on the paper tape. Each end-of-file mark on the magnetic tape is counted as one record when records are transferred. The number of records/files transferred is less than the specified number if the end-of-tape mark is sensed on the input magnetic tape before n is satisfied.

The total number of records/files is typed at the completion of the request.

Data Verification Requests

Card and Card

The card and card verify request format is:

VCC,u_1,u_2,x

Where: u_1 is card reader unit 1.

u_2 is card reader unit 2.

x is optional.

0 or blank	Format of data to be compared is 1700 formatted binary/ASCII.
1 to 99,999	Format of data to be compared is the 80-column binary card image.

The two card decks to be compared are input at card reader units u_1 and u_2. The IOUP program compares each card and, in the event of any discrepancy in the data, a diagnostic is printed. The comparison continues until an end-of-data on unit u_1/u_2 is detected. When verification is completed, the total number of records checked is printed (one card = one record).

Card and Paper Tape

The card and paper tape verify request format is:

VCP,u_1,u_2

Where: u_1 is the card reader unit.

u_2 is the paper tape reader.

The card deck to be compared against a paper tape is read by a format read at u_1 in the mode of the input deck. The paper tape is read by a format read in the same mode as the card deck at unit u_1. The data on one card is compared with one paper tape record. If a discrepancy exists, a diagnostic is typed. The verification proceeds until the last record on the card deck or paper tape, whichever is earlier, is processed. At the end of verification the total number of records checked is typed.

Card and Magnetic Tape

The card and magnetic tape verify request format is:

VCM,u_1,u_2,n,x

Where: u_1 is the card reader unit.

u_2 is the magnetic tape unit.

n is the number of records to be compared (1 to 99,999).

x is optional.

0 or blank	Format of data compared is 1700 binary/ASCII
1 to 99,999	Format of data compared is 80-column binary card image.

The card to be compared is read on unit u_1 until either n cards are read or the reader hopper is empty, whichever occurs first. This data is compared with each magnetic tape record until either an end-of-data or n is detected (actual number of cards/records of data is read, whichever occurs first). At the end of verification, the total number of records checked is printed. If any discrepancy exists, a diagnostic is typed.

Paper Tape and Paper Tape

The paper tape and paper tape verify request format is:

VPP,u_1,u_2

Where: u_1,u_2 are paper tape readers.

The two paper tapes are read consecutively in unformatted binary mode at unit u . A discrepancy in any record causes a diagnostic to be typed. At the end of verification the total number of records checked is typed.

Magnetic Tape and Magnetic Tape
Magnetic Tape and Magnetic Tape

The magnetic tape and magnetic tape verify request format is:

VMM,u_1,u_2,n

Where: u_1 is magnetic tape unit 1. } u_1 and u_2 can refer to the same unit.
u_2 is magnetic tape unit 2. }

n is the number of records of magnetic tape to be compared (1 to 99,999).

u_1 Equal to u_2 – Magnetic tape 1 on unit u_1 is read by a format read request in the mode of the data on the tape until the specified number of n records of data is read. If the end-of-tape mark is detected on tape 1 before n has been read, the number of records actually read is the number of records to be compared. An end-of-file counts as one record.

Magnetic tape 2 on unit u_2 is format read in the mode of the data of the corresponding record on tape 1 and compared for equality in the number of words and word-by-word meter in each record. A discrepancy causes a diagnostic to be typed. At the end of verification the total number of records checked is typed. Verification also ends when the end-of-tape mark is detected on tape before the specified number of records has been checked.

u_1 Not Equal to u_2 – The comparison of the data is one record from each of the two units. The verification ends when either the specified number of records are checked or the end of tape is reached on u_1 and u_2.

Magnetic Tape and Paper Tape

The magnetic tape and paper tape request format is:

VMP,u_1,u_2,n

Where: u_1 is the magnetic tape unit.

u_2 is the paper tape reader.

n is the number of records of magnetic tape to be compared (1 to 99,999).

The verification of data on the magnetic tape and paper tape is executed by reading one record of magnetic tape and comparing it with the corresponding record of paper tape. The comparison is for equality in the number of the words and a word-by-word match in each record. The comparison continues until either the number of words specified has been compared or the end-of-data on either unit is detected.

When a mismatch occurs, a diagnostic is output (refer to the MSOS Diagnostic Handbook).

Motion Control Requests

Advance Unit Number of Files

The advance unit number of files request format is:

TAF,u,n

Where: u is the magnetic tape unit.

n is the number of files to be advanced (1 to 4095).

Magnetic tape on unit u is advanced n number of files. When the end-of-tape mark is detected before all the n files advanced, the tape motion stops at the end-of-tape mark. A typed diagnostic indicates the total number of files advanced. An end-of-file mark is counted as one record.

Advance Unit Number of Records

The advance unit number of records format is:

TAR,u,n

Where: u is the magnetic tape unit.

n is the number of records to be advanced (1 to 4095).

Magnetic tape on unit u is advanced n number of records. When the end-of-tape mark is detected before all the n records are advanced, the tape motion stops at the end-of-tape mark. A diagnostic is typed to indicate the total number of records advanced. An end-of-file mark is counted as one records.

Backspace Unit Number of Files

The backspace unit number of files request format is:

TBF,u,n

Where: u is the magnetic tape unit.

n is the number of files to be backspaced (1 to 4095).

Magnetic tape on unit u is backspaced n number of files. If load point is detected before all the specified files are backspaced, tape motion stops at load point. A typed diagnostic indicates the number of files backspaced.

Backspace Unit Number of Records

The backspace unit number of records request format is:

TBR,u,n

Where: u is the magnetic tape unit.

n is the number of records to be backspaced (1 to 4095).

Magnetic tape on unit u is backspaced n number of records. An end-of-file mark is counted as one record. If load point is detected before all the specified records are backspaced, tape motion stops at load point. A typed diagnostic indicates the number of records backspaced.

Rewind Unit

The rewind unit request format is:

TRW,u

Where: u is the magnetic tape unit. The magnetic tape on unit u is rewound to its load point.

Write End-of-File Mark on Unit

The write end-of-file mark on unit request format is:

TEF,u

Where: u is the magnetic tape unit. An end-of-file mark is written on magnetic tape at unit u.

Set Density of Unit

The set density of unit request format is:

TSD,u,d

Where: u is the magnetic tape unit.

d is the density code.

0	Do nothing.
2	Select 200 bits per inch.
5	Select 556 bits per inch
8	Select 800 bits per inch.
16	Select 1600 bits per inch.

The density of magnetic tape on unit u is set to the specified density.

Unload Unit

The unload unit request format is:

TUL,u

Where: u is the magnetic tape unit.

EXTENDED DISK TO TAPE PROGRAM (EDTLP)

EDTLP is a save/load storage module drive disk pack program which executes as an ordinal using MSOS I/O requests.

The program is executed by manual interrupt of the computer and entering EDTLP and a carriage return. The program then responds with:

EDTLP IN
ENTER LOAD/SAVE

Response to this is SAVE for the disk to magnetic tape save operation or LOAD to restore the disk from a previously saved magnetic tape.

If SAVE is entered the program prints:

DEFAULT CONDITIONS
PHY UNIT MSOS LU LAST SA

The program then lists the physical unit, MSOS logical unit, and last used sector address for each file manager logical unit. If the file manager had not been called or is not in the system, the program prints:

****NO FILE MANAGER DATA AVAILABLE****

If the file manager files exist, then the program prints:

SPECIFY DEFAULT CHANGES
LL = MSOS LU (DEC)
SSSS = SECTOR ADDRESS (0001-7FFF)
(CR) = USE DEFAULT
LL,ssss

Where: ssss is the last sector to be saved; an input of 0000 means do not save this MSOS lu

If an invalid response is entered, one of the following is printed:

ll ERROR
ssss ERROR
FORMAT ERROR

The program then reprints:

ll,ssss

and waits for input. When completed, the program prints:

OUTPUT TAPE ON UNIT 0, CR WHEN READY

Respond with carriage return when ready. The program then prints:

UNIT	MSOS LU	SA
xx	yy	ssss

Where: xx is the physical logical unit

yy is the MSOS logical unit

ssss is the sector address

If the mass memory device does not respond properly to I/O commands, the following is printed:

MM I/O ERROR
xxxx hhhh y u zz

Where: xxxx is the sequence number of the record

hhhh is the sector address

y is the MM physical logical unit

u is the physical logical unit

zz is the MSOS logical unit

ENTER A FOR ABORT, C FOR CONTINUE, THEN CR

Where: A aborts the program

C flags the record as bad so it will not be written on MM during a LOAD, and continues the SAVE operation

If the program read request to MM fails, the following prints:

****MM FREAD I/O ERROR****

When an end-of-tape is reached, the program prints:

```
END-OF-TAPE, MOUNT NEXT TAPE, CR WHEN
READY
```

Respond with a carriage return when ready.

If the following message is printed, then an irrecoverable tape error has occurred:

```
****MT ERROR
```

The message EDTLP OUT at any time signifies exiting from EDTLP.

If the SAVE has executed to completion, the following is printed:

```
SAVE OPERATION COMPLETE
```

If LOAD is entered, the program prints:

```
INPUT TAPE ON UNIT 0, CR WHEN READY
```

Respond with a carriage return when ready.

The program prints the MSOS lu and last sector address:

```
UNIT    MSOS LU    SA
x       yy         ssss
```

Where: x is the physical logical unit

yy is the MSOS logical unit

ssss is the sector address

When end-of-tape is reached, the program prints:

```
END-OF-TAPE, MOUNT NEXT TAPE, CR WHEN
READY
```

Respond with a carriage return when ready.

Magnetic tape errors terminate the load with the printing of:

```
****MT ERROR
```

When a SAVE tape is generated, sequence numbers are generated for each tape record; if out of sequence, the program prints:

```
OUT-OF-SEQUENCE RECORD
```

Then the program requests instructions with the following:

```
ENTER DIGIT THEN (CR)

    1 - ABORT
    2 - CONTINUE
    3 - REREAD TAPE
```

If bad sectors were encountered, the program prints:

```
THE FOLLOWING SECTORS WILL NOT BE LOADED
xxxx-yyyy
```

Where: xxxx is the first bad sector

yyyy is the end-of-the-file sector

When the load operation has completed, the following message is printed:

```
LOAD OPERATION COMPLETE
```

WORD PROCESSING TEXT EDITOR (EDITOR)

This is a utility tape background program that creates, modifies, or retrieves files of data.

In this context, file refers to contiguous data at a specified place on mass storage (work file); this use of the word file is not to be confused with the more restricted definition of file as used by the job processor or the file manager. However, a job processor file may be placed in the work file and manipulated by the word processing text editor.

After calling the editor, file data may be entered into the work file in a number of ways:

- The LOAD statement loads ASCII data on a line-by-line basis, numbering each line as it is loaded into the work file.
- The GET statement loads a previously defined job processor file.
- The AUTO mode assigns line numbers in a preselected sequence but otherwise allows the operator to build a file as in the manual mode.

Manipulation of the work file is performed on a record-by-record basis. Since only one record is processed at a time, this makes text editing particularly applicable to:

- Editing plain English text (word processing)
- Editing programs (either FORTRAN or CYBER 18/1700 assembly language format) that are to be batch processed at a later date.
- Editing CYBER 18/1700 MSOS batch processor commands that are to be processed at a later time.
- Editing, merging, or rearranging other data that the user wishes to treat as a file

Since all work is done on the text editor's work file, source language programs treated as files are untouched by the text editor. This allows the user to save the old program or data until the new one is completely revised and checked.

In addition to manipulating the file data, the text editor has the ability to:

- Search for a data string (e.g., a statement label in one line or several lines)
- Format the data (i.e., align the spacing so that it corresponds to FORTRAN or assembly language line format; or to space output to listing device)
- Output the file on a specified device or as a Job Processor file

Since the text editor operates on a line-by-line basis, operator inputs (normally made from the comment device) are limited to an 80-character string. However, if the text editor is in manual mode, the operator must also enter the line number. In this mode, the text editor accepts a four-character line number followed first by a space and then by a data string up to 80 characters in length.

USING THE TEXT EDITOR

The text editor is a background utility program; to call it, the user must first place the system in job processing mode. (In the discussion that follows, ti is assumed that the operator is editing files from the comment device unless some other mode is specified). When the system replies with a J, the user enters an *EDITOR request. When the text editor is loaded, it replies with READY. The operator may then select (or create) his work file and manipulate it as he desires.

After each text editor command is executed, the editor indicates it is ready for further commands by displaying READY or > on the next line of the comment device. The operator types the next request on the same line and enters the complete request by pressing the RETURN key.

NOTE

The text editor works on a single file at a time. If two or more files are to be processed during a single call to the text editor, each of these files should be defined as a job processor file. The files may be read, processed, and rewritten to the job processor as described in the GET, MERGE, and SAVE commands.

To exit from the text editor, the user types EXIT and presses the RETURN key. At this point, the work file is released along with the scratch file. However, a copy of the file may have been transferred to the job processor files or listed on an output device for subsequent use.

The complete list of all text editor commands is shown in table 14-4. The typical command consists of two parts: a command type followed by a parameter list (i.e., COMMAND, parameter 1, parameter 2, . . .). It is not necessary to type the entire command type; a minimum abbreviation that distinguishes this command from all others is sufficient.

Trailing commas may always be omitted when one or more of the parameters separated by the commas are omitted.

Each of the commands is described in detail in the following section.

Parameters used in text editor commands are defined as follows:

ai	Member of character string
bi	Member of character string
fileid	File identifier
k	Text line number
lu	Logical unit
n	1. Line number 2. Incremental line value
x	Statement modifier

DETAILED DESCRIPTION OF TEXT EDITOR COMMANDS

Program Entry

The program entry command has the following format:

*EDITOR (cr)

No parameter list is supplied. The editor replies with READY on the next line. At this point, the program is in the manual data entry mode (described below). If that mode is not desired, one of the other commands must be used.

TABLE 14-4. TEXT EDITOR REQUESTS

Command (minimum abbreviation is underlined)	Parameters	Action
		Entry/Exit Program Control Statements
*EDITOR	None	Job processor passes CPU control to text editor.
EXIT	None	Text editor returns control to job processor.
CONTROL	,lu	Changes editor's command input to logical unit (lu)
		Entering or Changing Data in Work File
(manual)	None	Allows operator to enter one line of text. Operator must supply line number and text. Can be used anytime editor replies with READY or >.
AUTO	,n	Enters the user-specified line from the input device; line number is previous line plus n
LOAD	,lu,n	Loads data into work file from specified lu; sequences line number in increments of n
GET	,fileid,	Load job processor file named fileid into work file; sequences line number in increments of n

TABLE 14-4. TEXT EDITOR REQUESTS (Contd)

Command (minimum abbreviation is underlined)	Parameters	Action
MERGE	,fileid,n	Merges job processor file named fileid with the existing file in work file. New data is inserted following line n.
DELETE	,k or	Deletes line of text numbered k, or
	,k_1,k_2	deletes all text between (and including) lines k_1 and k_2
CHANGE	,*a_1. . .a_m* ,*b_1. . .b_m* ,k_1,k_2	Changes the character string $a_1 \ldots a_m$ to the new character string $b_1 \ldots b_m$ in all lines between (and including) k_1 and k_2
CLEAR	None	Clears the work file
		Search for or Save Data
SEARCH	,*a_1. . .a_m* ,k_1,k_2	Searches for character string in all lines between (and including) k_1 and k_2; lists line numbers where character string was found
LIST	,lu,k_1,k_2,x	Lists contents of all line numbers between k_1 and k_2 on logical unit (lu). If x is blank, also lists the line numbers (if lu is blank, comment device is used)
DUMP	,lu,k_1,k_2	Same as LIST except line numbers are not included, and list formatting commands cannot be used
SAVE	,fileid	Saves work file as a job processor file named fileid
		Data Formatting
RESEQ	,n_1,n_2	Resequences line numbers starting at line n_1. Value n_2 is added to that line number and to every line number following n_1.
ALIGN	,x,n_1,n_2	Aligns fields in file lines to conform to: • Assembly language format (x = A) • FORTRAN (x = F or x = blank) • Align data starting at line n_1 and terminating at line n_2 See AUTO or manual mode for command. Text is: • SNGL – Single space listed output • DOUB – Double space listed output • SPAC xx – Space down xx lines • PAGE – Go to top of form
ADF		
ADF	,lu,nnnn	Advance file
BSF	,lu,nnnn	Backspace file
ADR	,lu,nnnn	Advance record
BSR	,lu,nnnn	Backspace record
REW	,lu	Rewind logical unit
		lu is the logical unit nnnn is the number of iterations

After each text editor command is executed, the program reverts to manual data entry mode and indicates this with a READY or > reply.

NOTE

The READY or > reply occurs on the next line of the comment device. It is followed by a space. Any command should be entered on this same line and should be terminated by a carriage return.

Program Exit

The program exit command has the following format:

EXIT (cr)

No parameter list is supplied. This command may be used only after a READY or > reply. Control returns to the job processor.

NOTE

if the operator causes an exit from the text editor in an abnormal state and the editor is currently using job processor files, it is possible that all current job processor files being used by the text editor may be lost.

Change Editor Control Input

This command has the following format:

CONTRL,lu (cr)

This transfers input control from the current logical unit (normally the comment device) to the logical unit designated.

If the designated logical unit fails and MSOS signifies this on the comment device with a L,xx,FAILED,y reply, the operator's entry of the CU statement returns control to the comment device.

While operating from the specified logical unit, text editor replies are printed on the line printer.

Control remains with the specified logical unit changed by a subsequent CONTRL statement.

NOTE

When the text editor is called, all parameters are reinitialized. Specifically, the work file and scratch files are initialized, and operator interface control resides in the current system comment device.

Manual Mode Data Entry

No command is associated with this entry mode. As soon as the text editor replies with READY or >, the line of data can be entered in the form:

Where: nmbr is the line (record) number, which can be between 1 and 9999. If this number is already assigned to another line, the old data for that line is deleted and the newly entered data replaces it.

sp is space.

data is 80 characters (maximum). Error correction characters are excluded up to a total of 85 characters. If a carriage return is not found within the line buffer size, the text editor supplies a carriage return at the 86th character position.

Leading zeroes in the line number are suppressed and the number is right-justified in the first four character positions.

The editor signifies it is ready for second and succeeding manual line entries (or a new command) with a > reply. Following execution of any command except AUTO, the text editor defaults to manual mode condition. The first editor reply (which follows all commands except repeated use of manual mode entry) is READY.

Automatic Mode Data Entry

This command has the following format:

AUTO,n (cr)

Automatic entry is similar to manual entry except the text editor supplies the line number. The value n is the desired increment between line numbers. If text has been previously entered and the last line of text entered was m, then the starting line number is m + n. If no text has yet been entered in the work file, the starting line number is n. If n is omitted, the value of n is assumed to be 10. (In this case, the comma may also be omitted).

The text editor replies on the following line with a four-digit line number without leading zeros.

The operator types in the text desired on this same line (up to 80 characters), ending with a carriage return.

The text editor replies with the next line number, etc. To exit from the automatic sequencing mode, the operator types a carriage return instead of more text.

NOTE

In all text editor commands using n, both the n and the preceding comma may be omitted. In this case, n equals 10.

Error Recovery During Data Entry

If an error is discovered while typing a line before the carriage return has been pressed, and only a few characters have been typed since the error was made, the character ← (upper case letter O) may be typed once for each preceding character to be deleted. The correct character may then be typed. For example, suppose the following formula is to be typed:

Y=X**2+Z**2+ATAN(Q+.0134)

Instead the user types:

Y=Z**2+Z+AT

At this point the error is discovered. The user types the ← character three times, and then enters the correct characters. The typed line would appear as follows:

Y=X**2+Z+AT ← ← ← **2+ATAN(Q+.0134)

The above line would be interpreted by the text editor as the desired formula.

On conversational display terminals, the back-arrow cursor control key ← may also be used to delete characters. Each time the key is pressed the cursor is moved back one character. When the correct characters have been typed, the corrections appear on the screen in the exact format that the computer receives them.

If an error in a line is discovered before the carriage return has been pressed but after more than a few characters have been typed since the error occurred, the entire line may be deleted and retyped. To do this the procedure is as follows:

1. Press RUB-OUT
2. Press LINE FEED or cursor control
3. Press carriage return
4. Retype the line

The 90-character line buffer size limits the ability to correct text as described above. The total text line including the line number must be 85 or fewer characters (errors are carried in a separate buffer). Otherwise, the text editor truncates the data by supplying its own carriage return at the 86th character position.

Load File From Specified Logical Unit

This command has the following format:

LOAD,lu,n (cr)

This command loads ASCII data into the work file from the specified logical unit, and sequences the statement in increments of n (the default condition for n is n = 10, and the trailing comma may be omitted).

The loading operation terminates if an end-of-file (EOF) occurs or if the specified entry device fails. In the latter case, the operator responds to the failure message:

L,xx,FAILED,yy
ACTION

with a CU statement. Then in both cases, the text editor returns control to the comment device (or to the control device if a CONTRL statement previously redesignated the command input device).

The line number value must remain with the range 1 through 9999. If there is text already in the work file when LOAD is executed, the new data is appended to the existing file. The first new text line is numbered n + m where m is the last statement number of the previously existing work file number. The next statement numbers are m + 2n, m + 3n,... If an overflow occurs (m + kn > 9999), all lines (records) up to the record causing the overflow are saved in the work file. However, the loading operation terminates and the error message:

LINE NUMBER OVERFLOW

is displayed on the comment device.

(At this point the user may ordinarily resequence the existing file, and may then continue loading from the first overflow statement with a new LOAD statement. Alternatively, he may dump the existing file as a job processor file, clear the work file, and load the overflow portion in the reinitialized work file witha new LOAD statement).

Load File From Job Processor

The format of this command is as follows:

GET,fileid,n (cr)

This command is similar to LOAD, except the file loaded already is defined in the job processor with the label fileid (one to six alphanumeric characters beginning with a letter). The numbering parameter n designates:

1. Starting line number
2. The increment between line number values

Previously entered program text in the work file is automatically cleared; this command cannot be used to merge files as in the LOAD command case.

CAUTION

The file must be closed before entering the text editor.

Merge File from Job Processor

The format of this command is as follows:

MERGE,fileid,n (cr)

This command allows the operator to merge a job processor file into the file currently existing in the work file. The file to be merged is identified by the alphanumeric label fileid. This file is inserted in the existing file starting at statement n. Succeeding merged statements are numbered n + 10, n + 20, ..., n + k. If there are text lines in the original work file following the last fileid line (now numbered n + k), each of these lines has its value increased by (n + k).

If n is omitted, fileid file is appended to the last statement in the work file, numbered m. New lines from fileid are then numbered m + 10, m + 20, etc.

Delete Lines From File

The formats of this statement are:

DELETE,k (cr)

DELETE,k_1,k_2 (cr)

Where k, k_1, and k_2 are line numbers.

This command deletes the single line numbered k or deletes all lines between and including lines numbered k_1 and k_2.

If no parameters are entered, this command is rejected. The command is also rejected if k_2 is less than k_1.

Change Part of One or More Lines

The format of this command is as follows:

CHANGE,*a_1...a_n*,*b,...b_m*,k_1,k_2 (cr)

This command substitutes one character string (b_1...b_m) for another character string (a_1...a_n) in all statements between (and including) those numbered k_1 and k_2. The character strings a_1...a_n and b_1...b_m must be 20 characters or less. Each character string must start and end with a delimiter. The delimiter may be any legal alphanumeric character (it is * in the example) except a comma; however, it must be the same character at the start and end of each string. It must not be the same as any other character in the string, since the text editor treats each appearance of the delimiter character as a string delimiter and truncates the character string (and probably declares an invalid command) at that point. A comma cannot be a delimiter since it is required as a parameter delimiter.

If character string a_1...a_n is not found in the series of searched lines, no changes are made. The text editor informs the operator of this with the reply:

OPERATION FINISHED – STRING NOT FOUND

If the string a_1...a_n is found in one or more of the searched lines, the other string b_1...b_m is substituted whenever the original string is found. The text editor informs the operator by displaying at the comment device:

n CHANGES

Where: n is the number of lines in which the substitution occurred.

NOTE

If m is greater than n (i.e., the other string has more characters than the first string) and the original line length is close to 80 characters, the substitution occurs, but all characters beyond the new 80th character are lost.

If k_2 is omitted, only line number k is checked for the character string. If both k_1 and k_2 are omitted, the entire work file is searched for the a_1...a_n string. k_2 must be greater than k_1.

Clear the Work File

The format of this command is as follows:

CLEAR (cr)

This command clears the entire contents of the work file so that new text can be entered.

Search for a Data String

The format for this command is as follows:

SEARCH,*a_1...a_n*,k_1,k_2 (cr)

This command searches all lines from k_1 to (and including) k_2 for the string of characters designated a_1...a_n. As in the CHANGE command, the data string is set off by a delimiter (* in the example shown). The same delimiter restrictions described in CHANGE apply to SEARCH.

If k_2 is not specified, only line k_1 is checked. If neither k_1 nor k_2 is specified, the entire file is searched. k_2 must be greater than k_1.

Two results are possible. If none of the lines searched contain the data string, the text editor displays:

OPERATION FINISHED – STRING NOT FOUND

If the string was found in one or more of the lines searched, and if the string was found more than once in a single line, the line number of each occurrence is displayed:

STRING FOUND IN LINE

k_n
k_n
k_n
k_x
k_z
k_z

The string occurred three times in line k_n, once in line k_x, and twice in line k_z

List Lines of Text

The format of this command is as follows:

LIST,lu,k_1,k_2,x (cr)

This command lists all text lines between k_1 and k_2 on the logical unit specified. If no logical unit is specified, the comment device is used. If x is not a blank, no line numbers are included in the listing.

NOTE

If a display is used as the listing device and the number of lines requested is greater than the number of lines that can be concurrently displayed on the display screen, the listing takes the form of a scrolled output. Once the screen is initially filled, scrolling continues as each new line is added until the last line is displayed at the bottom of the screen. For the slow character display rate (300 characters per second), this can be a convenient way to inspect the file; for the rapid display rate (9600 characters per second), the user may wish to list only the number of lines that the screen displays concurrently. Note that this number may include blank lines as specified by the list formatting commands discussed below.

If k_2 is omitted, only k_1 is listed; if both k_1 and k_2 are omitted, the entire file is listed. If k_1 is larger than the line number of the last line in the file, only the last line is listed. k_2 must be larger than k_1.

The special list formatting commands are described later in this section.

Dump Lines of Text

DUMP,lu,k_1,k_2 (cr)

This command is similar to LIST except:

- Line numbers are not included
- There is no default lu, the editor uses the lu specified as its dump device
- List formatting commands do not apply: the dump is always single spaced and continuous
- A end-of-file is written to DUMP lu upon completion of transfer

The conditions for line number parameters k_1 and k_2 are the same as for the LIST command.

Save Work File as a Job Processor File

The format for this command is as follows:

SAVE,fileid (cr)

This command saves the work file as a job processor (background) file. Fileid is an alphanumeric consisting of six or few characters, with the leading character being a letter. Before saving the file, the user must:

- Define the file under the job processor. The command is *DEFINE,fileid,sc,date (see section 9).
- Be sure that no data is currently stored in the file and that the file was never opened as a write type of file. The file must currently be closed.
- When using the file outside of the text editor, the file may be opened only as a read file. If opened as a write file, the pseudo tape driver releases and redefines the file when an attempt to write into it is made. This, in effect, clears and then overwrites the data in the file.

The following example demonstrates use of job processor files as input and output for the text editor:

	Comments
*JOB	
J	
*DEFINE,AA,AA	Define input and output files in job processor
*DEFINE,BB,BB	
*OPEN,AA,AA,W,19	
*LIBEDT	
*T,10,A,19,A,2000	Open AA to receive 2000 ASCII lines; close AA
*Z	
*CLOSE,AA,AA	

```
*EDITOR
 READY GET,AA,1
 READY DELETE,1,1500        } Text edit file AA in
 READY RESEQ                  the work file
 READY AUTO
 .
 .
 .
 Additions made
 .
 .
 .
 READY SAVE,BB              } Save as file BB in job
 READY EXIT                   processor; exit

*OPEN,BB,BB,R19               Open and use as read-
                              only file for the user

User program or MSOS
utility

*CLOSE,BB,BB
*DEFINE,CC,CC                 Define a third job
                              processor file

*EDITOR
 READY GET,AA,1               Text edit data in AA,
 READY DELETE,50,2000         save in BB and CC as
 READY SAVE,BB                job processor files.
 READY SAVE,CC                Data that was in BB is
 READY EXIT                   lost and replaced with
                              the same data that was
                              stored in CC.
```

NOTE

When the file is saved as a job processor file, the line numbers are deleted.

Resequence Line Numbers

The formats for this command are as follows:

RESEQ (cr) or

RESEQ,n,n_1 (cr)

The first command shown resequences line numbers starting at the beginning line in the work file, with a line interval value of 10 (i.e., n, n + 10, n + 20. . . .).

The second command shown resequences line numbers starting at line n and using a line interval value of n_1 (i.e., n, $n + n_1$, $n + 2n_1$, . . .).

Align Text

The format for this command is as follows:

ALIGN,x,n_1,n_2 (cr)

This command aligns text in the format required by CYBER 18/1700 assembly language (x = A) or FORTRAN (x = blank or F). Starting line is designated by n_1, ending line is n_2. If n_1 is omitted, one is assumed; if n_2 is omitted, last line is assumed.

Assembly language text is aligned according to these rules:

- The label, operation, operand, and comment fields are aligned to start in character positions 5, 12, 17, and 31, respectively (corresponding to text positions 1, 8, 13, and 27, respectively).
- In the original unaligned text, each field is delimited by at least one blank. If there is to be no label field there must be at least one blank at the start of the line.
- If a field extends beyond the starting character position of the next field, the extended field is completely reproduced and the next field is separated from it by two blank characters.
- If the first non-blank character is an asterisk (*), the line is treated as a comment line and the asterisk and all following characters in the line are moved intact so that the asterisk is in character position 5 (text position 1).
- Any text that extends beyond the end character position (data text character position 80) after alignment is truncated and lost to the user.

An example of assembly language alignment by the text editor appears below.

Note that some job control statements may not be aligned properly with this request. Control statements should be added or corrected after alignment, as necessary.

Text Before Alignment:

```
 90 LDQ* XADD
100* CALCULATE STARTING ADDRESS
110 LDA =N96 WORDS/SECTOR
120 ADD =XBASE+REL1-REL2 ALLOW FOR ADDRESS
130 START NOP 0
```

Text After Alignment:

```
 90      LDQ*    XADD
100* CALCULATE STARTING ADDRESS
110      LDA     =N96   WORDS/SECTOR
120      ADD     =XBASE+REL1-REL2
                 ALLOW FOR ADDRESSES
130 START NOP  0
```

FORTRAN text is aligned according to these rules:

- If a C is the first text character in a line (that is, a C is in character position 5 including line number), the line is not altered.
- If an * is the first non-blank character of text in a line, the line is justified as a continuation line with the * in character position 10 (next character position 6).

- If a numeric string is the first non-blank character in a line, the string is assumed to be a statement label. The line is then justified so that the right-most character of the numeric string is in character position 5 (text character position 9). The label field must be delimited in the original unaligned text by the occurrence of any non-numeric character. Label fields greater than five digits are truncated. Any text that extends beyond end character position (data text character position 80) after alignment is truncated and lost to the user.

- If a non-blank, non-numeric character other than an asterisk is the first character in a line, the line is justified so that the first character is in character position 11 (text character position 7).

An example of text aligned according to FORTRAN format is shown below. Note that care must be taken not to begin a line with the character C if a comment line is not intended, as in line 1.

Text before Alignment:

```
50C(I)=0
60A(I)=0
70Q=Y+Z+LOG(X*X+35.4*X)
80*+Z**5
90 10 IF(I.EQ.L) GO TO 50
100 C(I)=1
```

Text after Alignment:

```
 50C(I)=0
 60       A(I)=0
 70       Q=Y+Z+LOG(X*X+35.4*X)
 80      *+Z**5
 90  10   IF(I.EQ.L) GO TO 50
100       C(I)=1
```

MOTION COMMANDS

The following motion commands allow the operator to control certain devices within the Text Editor.

ADF,lu,nnnn	Advance file
BSF,lu,nnnn	Backspace file
ADR,lu,nnnn	Advance record
BSR,lu,nnnn	Backspace record
REW,lu	Rewind logical unit

Where: lu specifies the logical unit. Care must be taken to ensure that the device interprets the motion command correctly

nnnn is the number of iterations; must not be greater than 9999

SPECIAL LIST FORMATTING

The special LIST commands are entered as normal lines of text, rather than as commands. This may be done either in manual mode or using the AUTO command. The LIST command interprets these special texts as they are defined below. It should be noted that only the LIST command uses these special text commands; the DUMP command ignores them as commands but treats them as ordinary lines of data (i.e., if the working file is dumped, these special texts appear as they were entered).

The special LIST commands are:

.SNGL – Single-space text. Note that the editor defaults to single spacing.

.DOUB – Double-space text. Editor double spaces until the .SNGL command is found.

.SPAC XX – Space down 1 through 99 lines (consecutive .SPAC XX commands are illegal).

.PAGE – This causes a top-of-form

NOTE

Since a check is made for these special words at the start of each line of text during the listing operation, no normal text line may begin with these mnemonics, even if more text follows the mnemonic.

TEXT EDITOR COMPATIBILITY REQUIREMENT

- The text editor is designed to operate under control of MSOS.

- Since the work file and scratch file reside on disk, the file manager is needed to assign file space.

- Two file numbers are used by the text editor. If these numbers conflict with file munbers reserved for an application program, the conflicting file number(s) must be reassigned in either the text editor or the other program (DEBUG or COSY may be used to accomplish the reassignment).

- Area 4 of allocatable core should be at least 2125 words in length to accommodate the file manager. If insufficient space is allocated, background processing may hang indefinitely.

- FORTRAN should be a part of the system if the ALIGN,F command is used.

Table 14-5 is an example that shows the text editor operating on two files entered from cards, and saved as three job processor files (after editing). The text of the input files and listed output are not shown.

Devices used in the following examples are:

lu	device
2	Punch
4	Comment device (conversational display terminal)
9	Line Printer
10	Card reader

ERROR MESSAGES

The text editor displays on the comment error device the set of messages shown in table 14-6.

ON-LINE TRACING PROGRAM (TRACE)

The on-line program, TRACE, interpretively executes user-defined machine code and prepares information listings pertaining to each instruction for output on the standard list device. The data area of the traced program is operated upon in the same manner as the program in normal execution.

TABLE 14-5. TEXT EDITOR EXAMPLE

Comment Device Statements and Replies	Significance
*EDITOR	Text editor called under job processor control
READY LOAD, 10, 2	Load file from cards; sequence in increments of two
L, 10 FAILED 14 ACTION RP	Device not ready, operator corrects condition, file loaded
READY SEARCH, *FINT6* STRING FOUND IN LINE 52 72 80	Following search, string FINT6 is found in lines numbered 52, 72, and 80 (three occurrences total)
READY CHANGE, *FINT6*, :FINT6A: 3 CHANGES	FINTA6A substituted for FINT6
READY SEARCH, :FINT6A STRING FOUND IN LINE 52 72 80	Substitution did occur in all three places
READY DELETE, 10, 13	Deletes all lines between and including 10 and 13
READY RESEQ	File resequenced with line numbers n, n + 10, n + 20 . . .
READY LIST, 9	File listed on the line printer
READY SAVE, PG1	File saved in predefined job processor file PG1
READY CLEAR	Text editor's file space cleared
READY CONTRL, 10	Control statements are read from card reader. These control statements include load, data statements, and return control to comment device.
READY SAVE, PG2 READY CLEAR	New file is saved in predefined job processor file PG2. Then text editor file is cleared.
READY AUTO	Auto mode: Text editor provides line number at intervals of 10. Operator enters program line by line.
10C THIS PROGRAM IS TO FUNCTION AS A SWITCHING NETWORK FOR A CYBER 18-20 20 CALL SWITCH 30 IX=100 40 ASWTCH = $2020 50C EXIT BACK TO CALLER 60	This label is illegal (seven characters)
READY LIST, 9	List new file on line printer, nonaligned
READY ALIGN	Align if FORTRAN format
READY LIST, 9	List new file on line printer, aligned incorrectly because of ASWITCH label
READY 40 ASWTCH = $2020	In manual mode, re-enter line 40 (deletes old line 40 and substitutes new line with correct ASWTCH label)
>ALIGN	

TABLE 14-5. TEXT EDITOR EXAMPLE (Contd)

Comment Device Statements and Replies	Significance
READY MERGE, PG2 50	New file is merged with PG2, starting at PG2 statement line 50
READY LIST, 9	Merged file listed on line printer
READY DUMP, 2	Merged file punched
READY SAVE, PG2	Merged file resaved at PG2 (This destroys former data in PG2)
READY 15 .SPAC 10 >41 .DOUB >52 .SNGL >53 .PAGE	In manual mode, provide formatting statements as text statements. If former statements had these line numbers, new lines are substituted, otherwise new (format) lines go at end of file formerly called PG2.
>SAVE, PG3	Save the formatted file at predefined job processor file PG3
READY CLEAR	Clear editor file
READY GET, PG2, 1	Retrieve PG2 and number each line consecutively, starting with line 1
READY LIST, 9	List renumbered PG2 on line printer
READY CLEAR	Clear text editor file
READY GET, PG3	Retrieve PG3, number lines in increments of 10, starting with line 10
READY LIST, 9, 10, 70	List lines 10 through 70 of PG3
READY EXIT J	Return CPU control to job processor

TABLE 14-6. TEXT EDITOR ERROR MESSAGES

Message	Command	Error Condition
DISK READ ERROR	Any	Work file and scratch file are kept on disk; each line is read as a separate record. Disk read or write error may occur during any read operation.
DIRECTORY READ ERROR	GET MERGE SAVE	The parameter fileid cannot be obtained while reading the job processor directory.
FILE NOT DEFINED	GET MERGE SAVE	The parameter fileid is not in the job processor directory.
FILE SPACE FULL	Any except EXIT CLEAR CONTROL	The file manager has run out of space to assign the text editor (work file or user file).
INVALID COMMAND	Any	Necessary characters at beginning of command mnemonic are erroneous, or a necessary parameter is omitted or wrong (e.g., k_2 is less than k_1).

TABLE 14-6. TEXT EDITOR ERROR MESSAGES (Contd)

Message	Command	Error Condition
INVALID LINE NUMBER	Any but EXIT CLEAR CONTROL SAVE ALIGN	The line number parameter (k or n) is greater than 9999.
LINE NUMBER OVERFLOW	LOAD MERGE GET AUTO RESEQ	Line number is greater than 9999. For all but RESEQ, the text editor saves in the work file all data up to the line causing the overflow. For RESEQ, the work file is lost.
NAME NOT UNIQUE	AUTO ALIGN CHANGE CLEAR CONTRL DELETE DUMP LIST LOAD SAVE SEARCH ADR BSR ADF BSF REW	The operator specified only the first letter of the command mnemonic (A, C, D, L, or S). At least the first two letters of these commands must be specified.
PROTECTED FILE	SAVE	User tried to save work file in a file defined by a protected program (pseudo tape)

TRACE resides on mass memory in the system's program library. A program that is to be traced is loaded with an *L command. The trace program is loaded and executed by an *TRACE command.

Upon initial entry to the trace program, a message is printed on the standard comment device, which requests the specification of parameters.

SPECIFY PARMS (ssss,llll,eeee,aaaa,qqqq,iiii,x,y)

Where:

ssss is the four-digit hexadecimal address at which tracing begins. If ssss, llll, and eeee are defined to be a value that is less than the address of the beginning of unprotected core, the values are assumed to be relative to the beginning of unprotected core.

llll is the four-digit hexadecimal address where the traced program begins.

eeee is the four-digit hexadecimal address at which tracing terminates.

aaaa is the initial hexadecimal value of the A register.

qqqq is the initial hexadecimal value of the Q register.

iiii is the initial hexadecimal value of the I register.

x If this parameter is defined to be L, printout of the instructions within a loop is suppressed. Otherwise, all instructions are listed.

y if this parameter is defined to be S, printout of the instructions within a subroutine is suppressed. Otherwise, subroutine instructions are listed.

A sample job deck is shown in figure 14-13.

The trace program cannot trace through the monitor. It must relinquish control to the program being traced when calls to the monitor, jumps to the dispatcher, or jumps and return jumps into protected core (i.e., entry points in the table of presets) are detected. Whenever a READ, WRITE, FREAD, or FWRITE request is detected, the trace program determines the completion address (if specified) and inserts a recall of the trace program at that location. If no completion address is defined, tracing continues at the location following the I/O request. In all other situations, the trace program has the responsibility of recalling TRACE if further tracing is to be performed. The trace program also interprets the code, $FFFF_{16}$, as the end of the trace command and gives control to the program being traced.

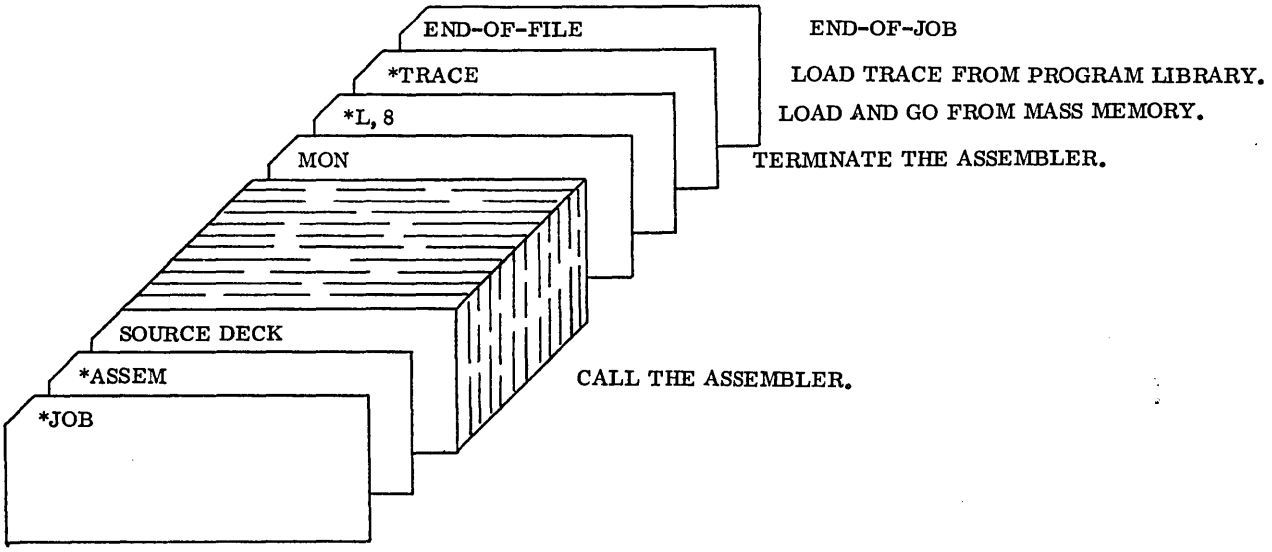


Figure 14-13. Sample Trace Job Deck

When the trace program detects a situation in which it cannot logically continue, the message

TYPE *C TO CONTINUE, TYPE *Z TO ABORT

is typed on the standard comment device. The *C response causes control to be given to the traced program and an *Z causes control to be relinquished to the job processor.

When the *C option is selected, it is implied that the traced program must recall TRACE to continue the trace listing. There are three recall entry points in TRACE which are accessed via a return jump (RTJ).

- TRACE1 – Continue tracing with current parameter.
- TRACE2 – Respecify parameters aaaa, qqqq, iiii, x, and y.
- TRACE – Respecify all parameters.

The listing that the trace program produces has eleven columns in a tabular form, with the following contents:

Code	Description
P	The current absolute value of the P register
RELA	The current P register value relative to the beginning of unprotected core.
CODE	The four-digit hexadecimal instruction contained at the address P
INST	The assembly language mnemonic for the current instruction
ADD	The effective address of the operand that is being operated upon by the current instruction
A	The current contents of the A register
Q	The current contents of the Q register
I	The current contents of the I register
P +1	The contents of the location at P +1
P +2	The contents of the location at P +2
CONT	The contents of the location where control is given by a jump or return jump instruction. The column is blank for all other instructions.

The following message appears on the standard list device wherever the trace listing is suspended:

```
EXECUTION TIME DURING THIS PART OF
EXECUTION
    1784-1**1774**1704**1784-2
    wwww  xxxx  yyyy  zzzz
```

Where: wwww, xxxx, yyyy, and zzzz are the hexadecimal count of the instruction time.

ABORT – To terminate a program when a condition (hardware or software) exists from which the program or computer cannot recover

ABSOLUTE BINARY PROGRAM – A program that must be loaded according to specific logical addresses

ABSOLUTE CODE – A code using absolute operators and addresses; a code using machine language

ABSOLUTE PROGRAM – A program composed of command sequence storage information, which may be loaded by a checksum loader

ADC – 1. Analog-to-digital converter 2. Address constant

ADT – Automatic data transfer a mode of data transfer on the CYBER 18 that simulates block transfer at the micro level via interrupts

AGENCY – A composition of processors dedicated to performing a single task

ALLOCATE – To reserve an amount of a resource in a computing system for a specific purpose

ALLOCATABLE MAIN MEMORY – That portion of main memory that can be assigned to programs by the core allocator (i.e., SYSDAT and resident program areas cannot be allocated)

ALTERNATE DEVICE – A peripheral device that can be assigned the tasks originally directed to another malfunctioning peripheral device

AMPLITUDE INACCURACY – 1. The relative amplitude error of analog values; the maximum absolute allowable error for the entire acquisition process (including cable transmission) is related to the amplitude peak value. 2. The accuracy a wave or alternating current value maintains during its maximum departure from its zero value

ANALOG CHANNEL – A channel that transmits an analog quantity (a voltage) rather than a binary value. The number of volts represents the value transmitted by the channel.

APPLICATIONS PROGRAM – A task or group of tasks that perform a defined function under the control of an executive system

A/Q CHANNEL – A CONTROL DATA 1700 Computer data channel, which can handle input/output only through the A register, is called the A/Q channel

ASSEMBLER – A computer program that generates machine instructions from symbolic input data by translating symbolic operation coding into computer operating instructions, assigning locations in storage for successive instructions, or computing absolute addresses from symbolic addresses. An assembler generates machine instructions from symbolic codes and produces, as output, nearly the same number of instructions or constants as were defined in the input.

ASSIGN – To reserve a part of a computing system for a specific purpose (usually refers to an active part such as an I/O device (e.g., tape unit)

ASYNCHRONOUS – Not synchronous; not happening, existing, or arising with a fixed-time correlation

ATP – Acceptance test procedures

AUTOLOAD – To place the resident routines of the operating system in core storage

AUTRAN-DACS – Automatic translator; a complete software system for either batch-sequencing or continuous process control, which can be configured, parameterized, and installed by the user. It is a flexible, English-like language that allows a process engineer to specify the process system and describe control actions conveniently. It can be intermixed with FORTRAN mathematical calculations. AUTRAN incorporates the parameterization of the integral data acquisition and control system.

BACK-UP STORAGE – Copies of permanent file images on tape (as generated by the disk-to-tape program)

BANK (MEMORY BANK) – A grouping of computer words into physically independent units which operate in parallel; the 1714 and 1784 can contain 16 banks, with 4096 words each.

BATCH – In MSOS, an object program running in a stacked job manner; shares the central processing unit with the priority program when a priority program is present and executes only when the priority program is not in control of the processor. Batch interrupts have lowest priority in the interrupt processing priority scheme.

BATCH JOB – A job submitted in the queue for batch processing (input queue)

BATCH PROCESSING – Pertaining to the technique of executing a set of computer programs so that each is completed before the next program of the set is started. Batch jobs are not considered to be time-critical since they do not need a particular response time (batch jobs will have the lower priority).

BDC – Buffered data channel

BENCHMARK – A point of reference from which measurements or comparisons for computer performance can be made

BIAS – A quantity added to the true exponent when packing a floating point number. Bias permits expression of both positive and negative exponents by positive numbers.

BLACK BOX – A generic term used to describe an unspecified electronic or mechanical device which performs a special function or in which known inputs produce known outputs in a fixed relationship.

BLOCK – 1. Consecutive matching words or characters considered or transferred as a unit, particularly applicable to I/O 2. Core: a unit size of core or MOS memory (4096 words)

BPI – Bits per inch. On multi-track magnetic tapes this is often used to mean frames per inch (fpi)

BUFFERED – Buffered drivers utilize the 1706 Buffered Data Channel. The 1706 controls read and write operations and moves data to or from core via the direct storage access bus (DSA).

BUFFERING – Overlapping execution of one or more I/O routines with the execution of the program that called them

CALIBRATION – Conversion of a quantity into measurable units (engineering units)

CARD COLUMN – A vertical line of punching positions on a card

CARD IMAGE – A one-to-one representation of the contents of a punched card; e.g., a matrix in which a 1 represents a punch and 0 represents the absence of a punch

CARD ROW – A horizontal line of punching positions on a card

CARTRIDGE DISK – One form of mass storage disk (1733-2/856)

CATENATE – To unite in a series, link together, chain

CC – Contact closure

CDT – Conversational display terminal

CENTRAL MEMORY – Refers to the directly addressable core storage of computers

CENTRAL PROCESSING UNIT – A unit of a computer that includes the circuits controlling the interpretation and execution of instructions; abbreviated as CPU.

CHAINING – A system for reading or writing records in which each record belongs to a list or group of records and has a linking field for tracing the chain

CHECKSUM – A summation of digits or bits used primarily for checking purposes and summed according to an arbitrary set of rules

CIRCULAR BUFFER – Refers to a buffer mechanism that allows write/read of data in a rotating manner; controlled by in/out and limit pointers

CLOSED LOOP CONTROL – A system capable of repeatedly reading data values from an object, comparing skew with desired values, and directly feeding back information into the object to correct value read

COMMON – An area of memory that may be shared between batch subprograms; common may not be preset with data.

COMPILER – A program that translates a programming language such as FORTRAN into an assembly language and often into machine language. A compiler may generate many machine instructions for a single symbolic statement.

COMPLETION PRIORITY – The priority assigned to the completion phase of a request (see request priority) after the requested task has been accomplished.

COMPONENT – A constituent part or ingredient; a software component is a basic logical software unit; several components form a module.

CONCENTRATOR – A device connecting a set of input lines with a set of output lines; the number of input lines normally is greater than the number of output lines.

CONTACT CLOSURE – A method of generating a singal by opening a closed electrical connection

CONTROLLER – A hardware device that controls access and data transfer to I/O units which are connected to it

CONTROL CARD, CONTROL STATEMENT – A command instruction recognized by the operating system

CONTROLWARE – Similar to firmware except that the control memory (e.g., micro memory) is self-modifiable.

CORE – 1. The core-type memory of the 1700 or System 17 or CYBER 18-10 CPUs. 2. Loosely used: The main memory (as opposed to the micro memory) of the CYBER 18-20 and CYBER 18-30 Timeshare systems. In fact, the core in these machines is composed of MOS memory elements.

CORE RESIDENT – The part of the operating system that resides permanently in central memory; it contains the code, various system tables, special buffers, etc. and begins at absolute location zero in the central memory

CORE SWAP – The contents of unprotected core is stored on mass storage and unprotected core is protected and made available for assignment by SPACE requests.

COSY – Program compression processor cumulate

CPU – Computer (central processing unit)

CREP – Core-resident entry point table. Holds entry points (linkage addresses) to protected programs executed in part 0 of core

CREP1 – Core-resident entry point 1 table. Holds entry points (linkage addresses) to protected programs executed in part 1 of core.

CR – Carriage return

CRT – Cathode ray tube

DACS – Data acquisition and control system; see AUTRAN-DACS.

DATA AREA – An area of memory that may be preset with data at load time and shared between subprograms; both batch and priority programs may have data areas.

DATA BLOCK – Equivalent to labeled common

DEADSTART – An initial action taken to start a computer when no software is resident or active on the system. This is serial interface via panel commands as contrasted to autoload.

DECK – A collection of punched cards that has a definite service or purpose; structured to represent a processing unit in the operating system

DESTRUCTIVE PROCEDURE – A procedure that is modified in place when executed. For example, a return jump to a subroutine modifies the entry point; therefore, the return jump and the subroutine are a destructive procedure.

DIAGNOSTIC – 1. Pertaining to the destruction and isolation of a malfunction or mistake 2. A message printed when an assembler, compiler, or monitor detects a program error

DIAGNOSTIC LOGICAL UNIT – A logical unit defined for diagnostic routines only

DIAGNOSTIC ROUTINE – A program or routine designated to locate and explain errors in a computer routine or malfunctions of a hardware component.

DICHOTOMY – A division into two subordinate classes; e.g., all zero and all nonzero.

DIGITAL CHANNEL – A channel that is transmitting a binary value rather than a voltage

DIGITAL INPUT SYNCHRONIZATION – The process by which digital data input operations are synchronized with external devices whose outputs change so that sampling is not done while they are changing.

DIGITAL-TO-ANALOG CONVERTER – A device that converts digital channel data to an analog signal

DIRECT DIGITAL CONTROL – A closed loop control system in which the output depends directly on input and computation (all in one frame time)

DIRECT STORAGE ACCESS – Method of accessing blocks of data directly in 1700 core memory by the peripheral equipment, without using the A/Q channel; abbreviated as DSA.

DISK – A magnetic storage device; the usual mass memory device

DISPATCHER – The portion of the monitor which locates the highest priority program awaiting execution and executes that program.

DMA – Direct memory access. CYBER 18 terminology same as direct storage access

DOCUMENTATION – The group of techniques necessary for the orderly presentation, organization, and communication of recorded specialized knowledge in order to give an unquestionable historical reference record for reasons for changes

DOUBLE BUFFERING – Two accessing elements that share a buffer space; e.g., processing data in one buffer while data is being input to an alternate buffer

DRIVE – A hardware device such as a tape drive or disk drive

DRIVER – A program whose main function is to perform a physical I/O transfer of data between one storage medium and another (e.g., between central memory and mass storage, between central memory and magnetic tape)

DSA – Direct storage access

DSKTAP – A disk-to-tape utility program.

DUMMY DRIVER – An I/O driver that processes requests to a non-existent peripheral device, usually by returning control directly to the requesting program

ECC – Error correction code. A special technique for reconstituting garbled data on certain disk systems.

EMULATOR – The 1700 emulator is a firmware component that allows the CYBER 18 hardware to function as an enhanced 1700 computer

END-OF-FILE – Information designating the termination point of data or of a program

END-OF-FILE INDICATOR – A signal supplied by an input or output unit that makes an end-of-file condition known to the routine or operator controlling the device

ENGINEERING LOG – Also called engineering file. A file for saving unrecoverable I/O error information. Data in the log is stored by logical unit. Each failure item has an identifying time tag associated with the failure status word.

ENT – Entry points to programs

ENTRY – 1. An entry point to any program 2. Initiator entry point to an I/O driver 3. Continuator entry point to an I/O driver 4. Timeout entry point to an I/O driver

EOB – End of buffer

EOF – End of file

EOP – End of operation

EOT – End of text, end of tape

EQUIPMENT – An interface between a data channel and a unit; a channel controller

ERS – External reference specification

ETX – End-of-text

EXECUTE – To carry out an instruction or perform a routine

EXECUTION – The process whereby the instructions contained in a program direct the activities of the central processing unit

EXT – Externals; entry points used by this program in other programs.

EXTERNAL INTERRUPT – An interrupt that occurs as a result of conditions within peripheral devices or their immediate interfaces; interrupts that occur as a result of conditions within a data channel are classified as external or internal, according to specifications set forth in the individual hardware system reference manuals.

FCR – Functional control register. An internal register on the CYBER 18 that allows the user to select operational machine modes and determine machine status.

$FFFF_{16}$ – -0; often used as a flag to indicate end of a table, list, etc., or to indicate the physical absence of an I/O device.

FIELD LENGTH – The number of central memory words that a program occupies

FIFO – First-in-first-out; a method of handling queued entries

FILE ORDINAL – A number equated to a mass storage file for the duration of the job

FIRMWARE – A physical electronic component in which a program resides that is incorporated in a product to provide a programmed mode of operation defining the product's functional characteristics. Firmware is not self-modifiable and is subject to change or modification only by physical modification or replacement.

FLOW – A general term used to indicate a sequence of events

FNR – Find-next-request routine. Used by I/O drivers to find next request queued to an I/O device.

FPI – Frames per inch. Sometimes called bits per inch (bpi)

FREAD
FWRITE – The formatted read and write requests.

GHOST INTERRUPT – An unsolicited interrupt from a peripheral device or an unused line

HALF-DUPLEX CHANNEL – A channel capable of transmitting and receiving signals, but only in one direction at a time

HANG-UP – When a request is unable to be completed because a peripheral device is not able to issue the necessary interrupt, the condition is called an I/O hang-up.

HEAD OFFSET – A mode for moving the magnetic read head on a desk slightly off track center line to compensate for data written on another disk, which disk had heads not perfectly aligned with the reading disk. Head offset is used when data cannot be read initially, in an effort to find some position of the head which can read the data.

HOLLERITH – A data code used by the COSY program

HOOK – Any piece of software that is embedded in the operating system, whose presence serves only to generate or save information about the activities of the operating system and whose presence in the operating system is not essential to and does not alter the functions of the operating system

HOUSEKEEPING – 1. Operations in a routine that do not contribute directly to the solution of a problem, but which are necessary to coordinate with the operation of the computer 2. Those necessary steps of computer operation that are common to nearly all instructions of a particular computer

ID – Identification

IFIPS – Internal Federation for Information Processing Societies

INDEX SEQUENTIAL – A method of file organization in which records are in a logical collating sequence, according to a key that is part of every record; a separate index or levels of indexes are maintained to give the location of certain records or segments of the file. The records may be accessed sequentially in a serial manner or directly in a random manner, through the index structure.

INPUT/OUTPUT – The bi-directional transmission of information between computer memory and peripheral devices

INSTRUMENTATION – Those components in a process control system that are not part of the computer system; i.e., the test stands and their related panels

INTERLEAVING – A technique in multiprogramming whereby segments of one program are inserted into another program so that the two programs can be processed simultaneously

INTERLOCK – 1. To ensure that only one process at a time can update something in a computer system (e.g., a system table) 2. The result of interlocking; the user can obtain an interlock on a table, allowing him exclusive access to that table.

INTERNAL INTERRUPT – An interrupt occurring as a result of conditions within the computer mainframe or immediate interfaces

INTERRUPT – 1. To stop a process so that it can be resumed at a later time 2. A break in the normal flow of a system or routine so that the flow can be resumed from that point at a later time; an interrupt is usually caused by a hardware-generated signal.

INTERRUPT MASK – A device for preventing interrupts of lower priority from interrupting the interrupt handler currently controlling the CPU.

INTERRUPTABLE PROCESS – A process that is composed of an interruptable processor and interruptable data; this process may be interrupted, the processor taken away and applied to another process, and later reinstated and run to completion without ill effects from the interrupt.

INTERRUPT STACK – A region in SYSDAT that holds information concerning programs that have been interrupted by a higher priority I/O task. Entries are processed on a last-in-first-out (LIFO) basis.

INTSTK – The interrupt stack

I/O – Input/output

IPS – Inches per second

JOB PROCESSOR – The background program executive

JOB TERMINATION – Those activities necessary to logically terminate job execution

K – Kilo; thousand

LATCHING RELAY – Refers to a type of relay with contacts that remain in their last position if power fails (a nonlatching type relay's contacts opens if power fails); in connection with contact closures, failsafe refers to a failsafe condition for the external equipment, i.e., the hardware that connects to the equipment must be selected in such a way that no harm will be done to the external devices in the event of a power failure.

LIBRARY – An organized collection of standard, checked-out programs, routines, and subroutines that can be used to solve any types of problems. There are two principal libraries in MSOS: a system library for foreground programs, a program library for background programs

LIFO – Last-in-first-out; a method of handling stacks.

LINKAGE – The interconnections between subprograms or between a main routine and closed subroutines; for example, the entry into a closed routine and the exit back to the main routine

LIST – A sequence of ordered data items in which special items or groups of items can have different meanings

LIST PROCESSOR – A routine or a set of routines working on a list

LOADING – The process of transferring a program from external devices to storage; in MSOS the relocatable loader transfers a relocatable program to the first sequential available positions in core.

LOCATION – A position in storage where a computer word can be stored and which is usually identified by an address

LOG TABLES – The logical unit tables (LOG1, LOG1A, and LOG2)

LOGICAL UNIT – A number that can be used to reference peripheral units

LSB – Least significant bit(s)

LU – Logical unit

MACRO – On the CYBER 18 macro describes those features applicable to the actual set of hardware (e.g., macro instructions, macro interrupt, macro memory)

MACRO ASSEMBLER – The program that compiles source language into 1700 machine language statements (ASSEM)

MACRO INSTRUCTION – An instruction in a source language that is equivalent to a specified sequence of machine instructions; usually a mnemonic instruction that a programmer can write in a source program to call for library or special routines.

MAIN MEMORY – The memory bank on a CYBER 18/1700 machine. It may be composed of core (17xx) or MOS memory (CYBER 18-20/30 Timeshare). Main memory does not include the micro memory (see below) of the micro processor.

MAN-MACHINE COMMUNICATION – Software components that establish communication between the operating system and the operators.

MASK – A machine word that specified which bits of another machine word are to be operated upon.

MASS STORAGE DEVICE – A disk or drum capable of storing large quantities of information; it can be randomly accessed. Data may also be stored and accessed by sectors (96 words) in formatted mode.

MASS-STORAGE-RESIDENT – A part of the system that resides on mass storage and which is brought into core when needed by the system

MASTER CLEAR – A switch that returns a computer or peripheral devices to initial conditions; abbreviated as MC

MC – Master clear

MEMORY PROTECT – Hardware and software that prevent batch programs from destroying foreground programs or operating system main storage

MI – Manual interrupt

MICRO – On the CYBER 18, describes those features applicable to the actual set of hardware (e.g., micro instructions, micro interrupt, micro memory)

MICRO MEMORY – The program emulation memory of the CYBER 18 CPUs. It contains both ROM and RAM memory and is not currently programmable by the user.

MIPRO – Manual interrupt processor; processes the operator generated manual interrupt.

MODULUS – An integer that describes certain arithmetic characters of registers, especially counters and accumulators, within a digital computer; the modulus of a device is defined by R^n for an open-ended device and $R^n - 1$ for a closed (end-around) device, where R^n is the base of the number system used, and n is the number of digital positions (stages) in the device. Generally, binary devices with modulus 2^n use twos complement arithmetic; devices with modulus $2^n - 1$ use ones complement.

MONITOR – The supervisory routine in an operating system that coordinates and controls the operation of user and system programs.

MOTION – The request to position an I/O drive (e.g. rewind magnetic tape)

MSB – Most significant bit

MSEC – Millisecond

MSOS 5 – The Mass Storage Operating System for CYBER 18/1700 CPUs

MULTIPLEX – 1. To interleave or simultaneously transmit two or more messages on a single channel 2. To utilize a single device for several similar purposes or to operate several devices in a time-sharing mode

MULTIPROGRAMMING – The interleaved execution of two or more programs, which stay in the same memory, by a single processing unit.

NAM – The program name block

NMONI – The monitor call

NONDESTRUCTIVE PROCEDURE – A procedure that is not modified in place when executed; see destructive procedure

NONREUSABLE PROCEDURE – A procedure that is destructive and noninitializing.

OBJECT LANGUAGE – The language that is the output of a given translation process; e.g., the language into which an assembler or compiler translates a source language

OCR – Optical character reader

ODEBUG – On-line debug package

OPEN LOOP – A loop used to control a repeated operation, but having no feedback for self-correcting action; contrast with closed loop

ORDINAL – 1. A number that specifies the relative order of an element (such as a word in a table in memory) within a collection of items (i.e., all the words of the table). 2. In assembly language coding, the ordinal of the first element in a collection is one.

ORIGIN – 1. The absolute address of the beginning of a program or block 2. In relative coding, the absolute address to which addresses in a region are referenced

OVERFLOW – The state of having too many entries (bits) for a register, fixed sized list, etc.

OVERLAY PROCESSING – A technique for processing a program whose total storage requirement for instructions exceeds available memory; the user divides the program into elements which are brought into core at different points of processing. When brought into core memory, an element of an overlay program may occupy the same storage locations as another element that was previously executed.

PARAMETER – 1. A variable that is given a constant value for a specific purpose or process. 2. A quantity in a routine that specifies a machine configuration, subroutines to be called, or other operating conditions.

PARAMETER LIST – A portion of a calling statement which defines all special values necessary to the call

PARITY – A method of checking data quality.

PART 0 – A user-defined block of contiguous memory extending from location 0 up to the location END0V4; part 0 must be large enough to include SYSDAT, SPACE, allocatable core, and system common.

PART 1 – Part 1 is the block of contiguous memory immediately following part 0 and extending to the highest available core location; it contains the monitor and drivers, and may contain the file manager, FORTRAN library, and partitioned core.

PARTITION – One of a number of predefined segments of a given area in main memory into which a mass-storage-resident program may be read and executed. (See Part 0, Part 1 and Allocatable Core).

PATCH – A temporary correction to a program

PHYSICAL DEVICE TABLE – A table containing necessary parameters for the I/O driver. Every logical unit has its own physical device table (PHYSTB).

PHYSTB – Physical device table

POSITIONING TIME – The time required for the access arm to move a selected track on a disk

POSTAMBLE – A group of special signals recorded at the end of each block on phase encoded tapes for the purpose of electronic synchronization

PREAMBLE – A group of special signals recorded at the beginning of each block on phase encoded tapes for the purpose of electronic synchronization

PRESET TABLE – A table of protected program entry points that can be used by unprotected programs.

PRIORITY – A scheme for determining that a routine or job is to be executed before another.

PRIORITY LEVEL – All programs are assigned a priority level, which determines the use of the central processor. The highest program priority is 15; the lowest is -1.

PROCESS – A process or process control application is a function external to the machine that is to be monitored or controlled by the 1700. Programs performing operations with respect to a process are referred to as process programs.

PROGRAM LIBRARY – Library of background programs. These can be relocatable binary or absolute (program files).

PROGRAMMING SYSTEMS REPORT – A form containing a listing of code to replace or to be added to a specified software component; form AA1901; abbreviated as PSR

PROTECTED MAIN MEMORY – A defined area of main memory storage in which each word of that area can only be accessed by words in that area, thus providing memory protection

PSEUDO DISK – A portion of disk addressed as if it were a separate disk

PSEUDO TAPE – A portion of memory which is treated and addressed as if it were a separate magnetic tape unit.

PSR – Programming systems report

PUSHDOWN POP-UP STACK – A list that is constructed and maintained so that the item to be retrieved is the most recently stored item in the list; i.e., last-in, first-out (LIFO)

QUEUE – A list of requests waiting to be processed; MSOS requests are ordered FIFO by priority level. Queued requests are usually threaded to one another throughout core; however, they may be restricted to a certain area, as in the scheduler stack.

READ – To transfer information from an external device to internal storage

REAL-TIME – Pertaining to a program for which time requirements are particularly stringent

RE-ENTRANT – Programs that may be interrupted, called by interrupting programs, and resumed at the point of interruption without loss of continuity. A program may be re-entrant to any level; an interrupted program might be called again, etc.

RE-ENTRANT CODE – A code that does not alter itself during execution. The same body of code may be used concurrently by two or more processors. This feature saves space as does a serially reusable subroutine. It also saves time because there is no waiting. Re-entrant subroutines rely quite heavily on the use of registers especially for use in addressing so that each task will have its own data storage area and so that all valuable information will be stored if the processor is interrupted.

RELOCATABLE BINARY SUBPROGRAM – A program that can be loaded contiguously into available memory with the aid of a loader program

RELOCATABLE PROGRAM (OBJECT DECK) – A program that includes control information regarding program name, entries, externals, transfer address, and command sequence storage; it may be loaded anywhere in absolute form by a relocating loader.

REMOTE – Physically displaced; e.g., a remote terminal might be located miles from a central computer but be linked by telephone lines

REQUEST PRIORITY – The priority of a request with respect to other requests; determines when the request is processed

RESPONSE TIME – The time interval between the occurrence of an event and the perception of some action at the source of the event

RETURN – To transfer control back to a point in a program or subprogram from which a call was issued

REWIND – To return a tape or disk file to its beginning

RMS – Rotating mass storage; refers to disks and drums

ROW BINARY – Pertaining to the binary representation of data on punched cards in which adjacent positions in a row correspond to adjacent bits of data; e.g., each row in an 80-column card may be used to represent 80 consecutive bits of two 40-bit words

RSM – Request system modification

RTJ – Return jump

RUN-ANYWHERE – Programs that execute properly regardless of where they are executed in core memory; all addressing internal to the program is referenced by relative addressing.

RW – The read/write request processor

SEALING – Changing the value of a quantity by a factor in order to bring its range within prescribed limits

SCHEDULER STACK – The list space in SYSDAT that contains all the queued scheduler calls. Six queues are threaded through the space: one for programs that can be immediately executed; four for programs to be delayed, and one for empty entries. Programs in the scheduler queue are FIFO within priority; programs in each of the four timer queues are threaded on a time-to-go-before-execution basis.

SCHSTK – The scheduler stack

SCMM – Small Computer Maintenance Monitor

SECTOR – A contiguous space on mass storage, 96 words in length for MSOS.

SECTOR MARKS – Flags on disk tracks, marked off in equidistant points for addressing purposes.

SEEK – The process of moving the disk heads over the desired track.

SEQUENTIAL FILE ACCESS – A process for obtaining information from or placing information into a file where the access time depends on the number of undesired logical records that must be processed before reaching the desired location in the file (also referred to as serial access); a file on magnetic tape can only be accessed sequentially.

SERIAL RECORDING – Consecutive recording of bits of data on a single path (track)

SET POINT – Data output that informs the test object as to what reference point it should start and maintain

SI – System initializer

SIGNAL CONDITIONING – The transformation of an analog signal so that it can be processed by an A/D converter

SLEW – To pass data until desired end of input pattern is sensed.

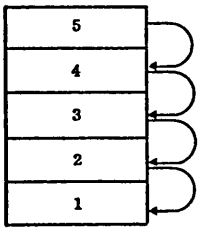
SMM17 – System Maintenance Monitor for the 1700 computer systems

SNAPDUMP – A selective dump performed at various points in a machine run

SOURCE LANGUAGE – Input language for a given translation process

SPOOLING – A technique of transferring jobs and data from one input device to another (usually a mass storage file) for processing later

STACK – A contiguous space for fixed sized entries. In a pushdown pop-up stack all entries must be contiguous, as shown below:



STATUS – A state or condition of hardware or a task; e.g., busy or not busy

STX – Start of text

SUBPROGRAM – A part of a larger program that can be coverted into machine language independently

SUBROUTINE – 1. A portion of a routine that causes a computer to carry out a well-defined mathematical or logical operation 2. A routine arranged so that control may be transferred to it from a master routine and, at the conclusion of the subroutine, returned to the master routine; such a subroutine is usually called a closed subroutine.

SYSTEM – A regularly interdependent group of subsystems forming a unified whole

SYSTEM FILES – The entire operating system as it appears on the system device; sometimes called the library

SYSTEM TABLES – Tables that are used by the operating system and which lie outside the user's field length (SYSDAT)

THREAD – A list of entries (requests) that each contain a pointer to the next entry; e.g., logical unit thread. Threads may extend throughout core. The position of a request in the thread determines how many requests are queued in front of it.

THROUGHPUT – The productivity of a computer based on all aspects of an operation; throughput of computers is often compared by calculating the amount of time required by each computer to complete the same processing.

TIME-SHARING – The capability of a computing system to accommodate more than one user during the same interval of time without apparent restriction caused by the existence of other users; a given device is used in rapid succession by a number of other devices, or various units of a system are used by different users or programs. The sharing is controlled automatically and may or may not include a priority scheme by using multiprocessing. The time-sharing may reduce total processing time from that required to do batch processing.

TRANSDUCER – A device for converting energy from one form to another

UNIT – A peripheral device capable of storing, receiving, transmitting, or interpreting data; connected to an equipment

UNIT RECORD DEVICES – Devices such as the card reader, line printer, and card punch

UNLOAD – To remove a tape from ready status by rewinding beyond the load point; the tape is then no longer under control of the computer.

UNPROTECTED – A defined area of core storage in which memory references are restricted to other locations in that area; memory accesses which reference a protected area will generate an internal fault condition, thus providing memory protection. References from a protected area may legally access this area.

UPDATE – 1. To modify a file with current information according to a specified procedure 2. To modify an instruction so that its operand address is changed by a stated amount each time the instruction is performed

USER – A programmer, process, or job that uses MSOS

USER PROGRAM – An object program loaded and entered under MSOS control; includes batch and priority programs and library routines

USER'S EXECUTABLE CODE – That portion of a program that represents steps that the computer will perform; for example, in FORTRAN, the GO TO statement results in executable code.

USER'S WORKING AREA – That portion of a program that results in storage for data prior to post-processing; for example, in FORTRAN, the COMMON statement generates a working area.

UTILITY ROUTINE – A routine in general support of the operation of a computer; e.g., a code updating, I/O, diagnostic, tracing, or monitoring routine

VOLATILE STORAGE – Temporary storage area to save register contents or data; used by the 1700 Monitor

WORST CASE – That which gives maximum stress or consumes maximum time; e.g., the pattern of ones and zeros in storage that creates the greatest noise or the maximum possible time between two significant programming operations

WRITE – To transfer information, usually from internal storage, to an output device

The area of core from 0 - FF_{16} is used as a communications area because it can be addressed directly by a one-word instruction. Its contents are defined in table B-1; all locations are protected except as noted.

An extended communications region table is provided to increase storage for essential system information. The table that resides in SYSDAT is accessed through core location E9. These locations in the table have been reserved for use. Any information in the table can be manipulated or used by using the contents of E9 and an index register set equal to the sequential number of the required item in the table. For example, if the contents of the fifth word of the extended table were required, it could be obtained by the following sequence of code:

```
ENQ      5
LDA-     ($E9), Q
```

The current table usage is shown in table B2.

TABLE B-1. COMMUNICATIONS AREA CONTENTS

Location	Label	Contents	Hexadecimal Equivalent
0		0001100011111111	18FF
1		0	0
2	LPMASK	0 0	0
3	ONE	0 01	1
4	THREE	0 011	3
5	SEVEN	0 0111	7
6		0 01111	F
7		0 01 1	1F
8		0 01 1	3F
9		0 01 1	7F
A		0 01 1	FF
B		0 01 1	1FF
C		0 01 1	3FF
D		0 01 1	7FF
E		00001 1	FFF
F		0001 1	1FFF
10		001 1	3FFF
11		01 1	7FFF
12	NZERO	1 1	FFFF
13		1 10	FFFE
14		1 100	FFFC
15		1 1000	FFF8
16		1 10000	FFF0
17		1 10 0	FFE0
18		1 10 0	FFC0
19		1 10 0	FF80
1A		1 10 0	FF00
1B		1 10 0	FE00
1C		1 10 0	FC00
1D		1 10 0	F800
1E		1 10 0	F000
1F		1110 0	E000
20		110 0	C000
21		10 0	8000
22	ZERO	0 0	0000
23	ONEBIT	0 1	1
24	TWO	0 10	2
25	FOUR	0 100	4
26	EIGHT	0 1000	8
27		0 10000	10
28		0 10 0	20
29		0 10 0	40
2A		0 10 0	80
2B		0 10 0	100

TABLE B-1. COMMUNICATIONS AREA CONTENTS (Contd)

Location	Label	Contents	Hexadecimal Equivalent
2C		0 10 0	200
2D		0 10 0	400
2E		0 10 0	800
2F		0 10 0	1000
30		0 10 0	2000
31		010 0	4000
32		10 0	8000
33	ZROBIT	1 10	FFFE
34		1 101	FFFD
35		1 1011	FFFB
36		1 10111	FFF7
37		1 101 1	FFEF
38		1 101 1	FFDF
39		1 101 1	FFBF
3A		1 101 1	FF7F
3B		1 101 1	FEFF
3C		1 101 1	FDFF
3D		1 101 1	FBFF
3E		1 101 1	F7FF
3F		11101 1	EFFF
40		1101 1	DFFF
41		101 1	BFFF
42		01 1	7FFF
43	FIVE	0 101	5
44	SIX	0 110	6
45	NINE	0 1001	9
46	TEN	0 1010	A
47 - B2		Reserved for user applications	
B3		Logical unit number of scratch unit	
B4		Top of thread of empty entries in schedule stack (TOMPT)	
B5		Location of FNR	
B6		Address of complete request subroutine used by drivers	
B7		Address of MASKT	
B8		Core location of next open location in interrupt stack (COUNT)	
B9		Address of request exit	
BA		Address of volatile storage release routine	
BB		Address of volatile storage assignment routine	
BC		Address of absolutizing routine for logical unit	
BD		Address of S parameter absolutizing routine	
BE		Address of C parameter absolutizing routine	
BF		Address of N parameter absolutizing routine	
C0		Most significant bits of the first scratch area sector number	
C1		Least significant bits of the first scratch area sector number on the library unit	
C2		Logical unit number of the library unit	
C3		Most significant sector number of first program library directory block (always zero)	
C4		Least significant sector number of first program library directory block	
C5 - E3		Reserved for FORTRAN (unprotected)	
E4		Used for load-and-go sector (unprotected)	
E5		Reserved for FORTRAN (unprotected)	
E6		Length of system library directory	
E7		Index to first mass storage entry in the system directory	
E8		Real-time clock counter, incremented once each timer interrupt	

TABLE B-1. COMMUNICATIONS AREA CONTENTS (Contd)

Location	Label	Contents Contents	Hexadecimal Equivalent
E9		Core address of extended core table	
EA		Location of the dispatcher	
EB		Core location of beginning of the system library directory	
EC		Temporary highest unprotected location +1	
ED		Temporary lowest unprotected location -1	
EE		Used by the job processor for returns from loader	
EF		Current priority level	
F0		Core location of next available volatile storage	
F1		Length of the table of presets	
F2		Location of the table of presets	
F3		Location of the breakpoint program when in the core (unprotected)	
F4		Location of entry for system requests	
F5		Largest core location used	
F6		Highest unprotected location +1	
F7		Lowest unprotected location -1	
F8		Address of internal interrupt processor	
F9		Logical unit number of standard input device	
FA		Logical unit number of the standard binary output device	
FB		Logical unit number of the standard print output device	
FC		Logical unit number of the output comment device	
FD		Logical unit number of the input comment device	
FE		Location of the common interrupt handler	
FF		Memory index register I (unprotected)	

TABLE B-2. EXTENDED COMMUNICATIONS REGION TABLE

Label	Operator	Address	Word	Comments
	ENT	MPFLAG		
	ENT	MAXSEC		
	EXT	JFILV4		
	EXT	RCTV		
	EXT	END0V4		
	EXT	DATBAS		
	EXT	SECTOR		
	EQU	CSYLST (9)		
	EQU	CSYINP (10)		
	EQU	CSYPUN (11)		
	EQU	JFILV4 ($7FFF_{16}$)		
EXTBV4	ADC	0	00	Mode switch: 0 = 32K 1 = 65K
	ADC	CSYINP	01	Standard COSY input logical unit number
	ADC	CSYPUN	02	Standard COSY output logical unit number
	ADC	CSYLST	03	Standard COSY list logical unit number
	ADC	0	04	First sector LSB of system core image
	ADC	0	05	First sector LSB of sector availability table
	ADC	0	06	First sector LSB of CREP† table (part 0)
	ADC	0	07	First sector LSB of CREP1 table (part 1)
	ADC	JFILV4	08	First sector LSB of job file directory

†CREP = core-resident entry point

TABLE B-2. EXTENDED COMMUNICATIONS REGION TABLE (Contd)

Label	Operator	Address	Word	Comments
	ADC	RCTV	09	Address of RCTV table in the monitor
	ADC	0	10	Unprotected core flag: 0 = Part 0 1 = Part 1
	ADC	0	11	Unprotected swap allowed: 0 = yes 1 = no
	ADC	AYERTO	12	Address location containing the year
	ADC	AMONTO	13	Address location containing the month
	ADC	ADAYTO	14	Address location containing the day
	ADC	END0V4	15	Last address of part 0 core
	ADC	0	16	First address of blank (system) common
	ADC	DATBAS	17	First address of labeled common
	ADC	0	18	COSY driver current physical device table address
	ADC	0	19	Job table initialization flag
	ADC	0	20	Mass memory location of engineering file
	ADC	SECT1	21	MSB of maximum scratch sector
MAXSEC	ADC	SECTOR	22	LSB of maximum scratch sector
	ADC	SECT3	23	MSB of maximum library sector
	ADC	SECT4	24	LSB of maximum library sector
	ADC	0	25	Last address of labeled common
	ADC	N16KMM	26	Number of 16K memory increments
MPFLAG	ADC	EXTSTK	27	Pointer ot extended interrupt stack
	ADC	LOG1A	28	Address of LOG1A table

The physical device tables are included in SYSDAT (the system and parameters program).

PHYSICAL DEVICE TABLE

Each device has a physical equipment table that contains the interfacing information specified by the user to the device. It contains the entry addresses to the driver responsible for operating the device, the station address that tells the driver which device to use, and the information which allows the driver to fulfill the current request. The table contains at least 16 words for a device. Words 0 through 15 have a standard function for all devices. Additional words are added for use by the output message buffer package and special use by drivers. Drivers written in kernal form have an additional eight specified words (words 16 through 23). Additional words for these kernal drivers begin at word 24.

Table C-1 gives a detailed description of each of the words in the basic driver physical device table (figure C-1).

TABLE C-1. PHYSICAL DEVICE TABLE WORDS

Word	Name	Description
0	ELVL	$520x_{16}$; a scheduler request to operate the driver initiator address at level x, the driver priority level
1	EDIN	The driver initiator address
2	EDCN	The driver continuator address; control is transferred to EDCN on interrupt at the priority level assigned to the interrupt trap region. This priority level must be the same as the priority level specified by word 0.
3	EDPGM	The driver error routine address; control is transferred to EDPGM when the diagnostic clock is counted down to negative by the diagnostic timer at the driver priority level.
4	EDCLK	The diagnostic clock; this location is set by the driver and counted down by the diagnostic timer for a hardware completion interrupt. It is set idle (-1) by a complete request.
5	ELU	The logical unit currently assigned to the device; zero if the device is not in use. It is set by the request processor and may be reassigned by find-next-request, and cleared by find-next-request or complete request.
6	EPTR	Call parameter list location for current request; it is set by find-next-request.
7	EWES	Hardware/Address Bits — Code 0 through 6 — Station 7 through 10 — Equipment 11 through 15 — Converter The equipment status is obtained by loading this word into Q, followed by input. Status is saved in word 12, ESTAT2.
8	EREQST	Request Status Bits — Code 15 — 1 if operation is in progress; 0 if operation is complete 14 — 1 if driver detects I/O hardware failure

TABLE C-1. PHYSICAL DEVICE TABLE WORDS (Contd)

Word	Name	Description
8	EREQST	Request Status Bits — Code 13, 12, 11 — The equipment class Code — Device 0 — Class undefined 1 — Magnetic tape 2 — Mass storage 3 — Card 4 — Paper tape 5 — Printer 6 — Teletypewriter 7 — Reserved for future use 10 through 4 — Equipment type constant (T) 0 — 1711 1 — 1721/1722 2 — 1723/1724 3 — 1752 4 — 713-10/711-100/713-120 5 — 1738/853 6 — 1751 7 — 1739-1 8 — 1738/854 9 — 1731/601 10 — Software buffer 11 — COSY driver 12 — 1728/430 13 — Core allocator 14 — 1733-1/854 15 — 1733-1/856-2 16 — 1733-2/856-4 17 — 1742-30 18 — 1742-120 19 — 1740/501 20 — 1732-2/615-73 21 — 1732-2/615-93 22 — 1732-1/1706/608 23 — 1726/405 24 — 1732-1/608 25 — 1732-1/609

TABLE C-1. PHYSICAL DEVICE TABLE WORDS (Contd)

Word	Name	Description
8	EREQST	Request Status Bits 10 through 4 Equipment type constant (T) Code — Device 26 — 1713 Keyboard 27 — 1713 Punch 28 — 1713 Reader 29 — 1729-2 30 — 1732-1/1706/609 31 — Software Dummy 32 — 364-4/361-1 33 — 364-4/361-4 34 — 1742-1 35 — 1777 Reader 36 — Pseudo Tape 37 — 1777 Punch 38 — 1729-3 39 — 1733-1/853 40 — 1731/1706/601 41 — 1726/1706/405 42 — 1747 43 — 1744/274 44 — 1536 (Local) 45 — 1501 (Remote) 46 — 1536 (Remote) 47 — 1544 (Remote) 48 — 1553 (Remote) 49 — 1555 (Remote) 50 — 1566 (Remote) 51 — 1547 (Remote) 52 — 1595 Serial I/O Card 53 — 1732-3/616/72 54 — 1732-3/616-92/95 55 — 1743-2 56 — 1745/210 57 — 1725-1 Card Punch 58 — 1720-1 Reader 59 — 1720-1 Punch 60 — Mag Tape Simulator 61 — 1732-3 Long Record Driver

TABLE C-1. PHYSICAL DEVICE TABLE WORDS (Contd)

Word	Name	Description		
8	EREQST	Request Status		
		Bits		
		10 through 4	Equipment type constant (T)	
			Code	Device
			62	1811-1 LIAT CRT/Printer
			63	1829-30/60 Card Reader
			64	1827-30/65119-1 Line Printer
			65	1860-72 Mag Tape
			66	1860-92/95 Mag Tape
			67	1832-5 Cassette Tape
			68	1833-5 Flexible Disk
			69	1833-1/1867-10 Disk
			70	1833-1/1867-20 Disk
			71	Extended Core Driver
			72	Pseudo Disk
			73	1843-2 Eight-Channel CLA
			74	1866-14 Cartridge Disk
			75	1866-12 Cartridge Disk
			76	1827-7 Matrix Printer
			77-89	Reserved
			90	Reserved - OCR
			91	915 Page Reader
			92	929 Document Reader
			93	936 Document Reader
			94	Reserved - OCR
			95	955 Page/Document Reader
			96	Reserved
			97	979 Reader/Sorter
			98	Reserved - OCR
			99	Reserved - OCR
			100 - 127	For user assignment
		3	Not used	(reserved)
		2	1	If device may be written by unprotected programs
		1	1	If device may be read from unprotected programs
		0	1	If device is not available to unprotected programs
9	ESTAT1	Status word number 1		
		15	1	If error condition and/or end-of-file is detected
		14	1	If fewer words are read than requested Driver
		13	1	If device remains ready after detecting an error or end-of-file or both

TABLE C-1. PHYSICAL DEVICE TABLE WORDS (Contd)

Word	Name	Description
9	ESTAT1	Status word number 1 Bits / Code / Device 12, 11, 10, 9, 8, 7, 6: Reserved for special use by individual drivers 5: 0 — If this is a control character 4: 0 — If this is the first character 3: 1 — If ASCII; 0 if binary mode — FNR 2: Reserved for special use by individual drivers 1: 1 — If format READ or WRITE; 0 if unformatted — FNR 0: 1 — If WRITE; 0 if READ — FNR
10	ECCOR	The location where the driver will next store or obtain data; it is set initially by FNR and updated by the driver (refer to Find Next Request, section 2).
11	ELSTWD	Last location +1 where the driver is to store or obtain data to satisfy the request.
12	ESTAT2	Status word 2; the last value of the equipment status (refer to word 7)
13	MASLGN	The length of the driver for this device when the driver is mass-storage resident. This word is zero if the driver is core-resident.
14	MASSEC	Contains the name associated with the sector number on mass storage; if the driver is core resident, this name is patched with $7FFF_{16}$.
15	RETURN	Used for a return address by NFNR, MAKQ, and NCMPRQ
16		These words may be added to the device table if required for special purposes for a particular driver. For example, they can be used to count the lines per page of output or to link several tables together all using the same driver.

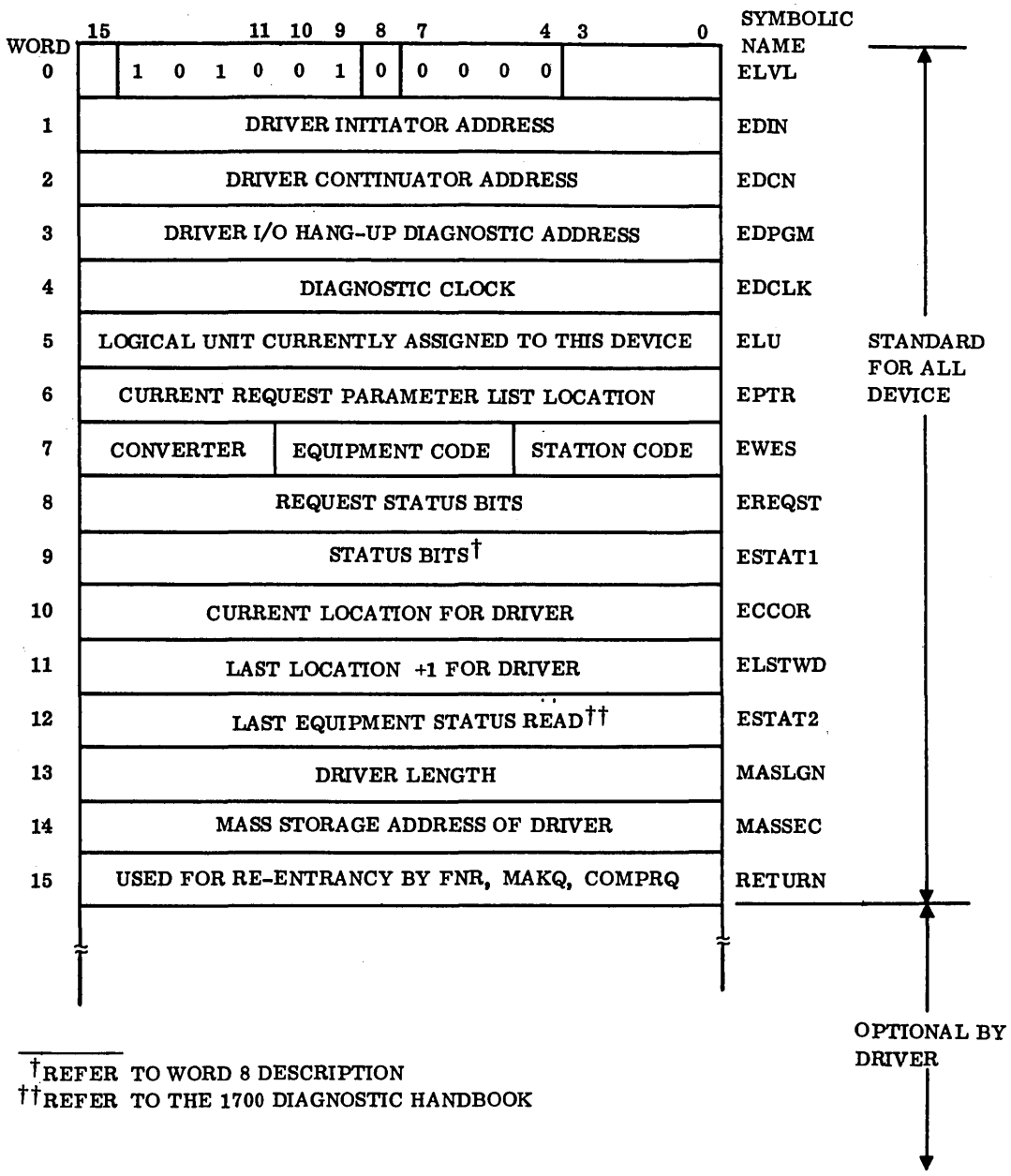


Figure C-1. Physical Device Table

The following information gives a detailed description of the words in the kernel driver's PHYSTB expansion (figure C-2).

Word 16 FLTCOD
Kernel fault code if an error occurs.

Word 17 DIAGLU
Diagnostic logical unit. If ELU equals DIAGLU, then the request is completed with error. The error is not logged in the engineering file and ALTDEV is not called.

Word 18 GHOSTI
A count of the number of ghost (unexpected) interrupts that have occurred.

Word 19 MICROI
The device's micro-interrupt number, if any.

Word 20 TIMOUT
The amount of time in seconds to wait for an expected interrupt.

Word 21 SENTRY
The status after the initial entry into the kernel.

Word 22 SINTER
The status after the device has interrupted (entry into the kernel's continuator entry).

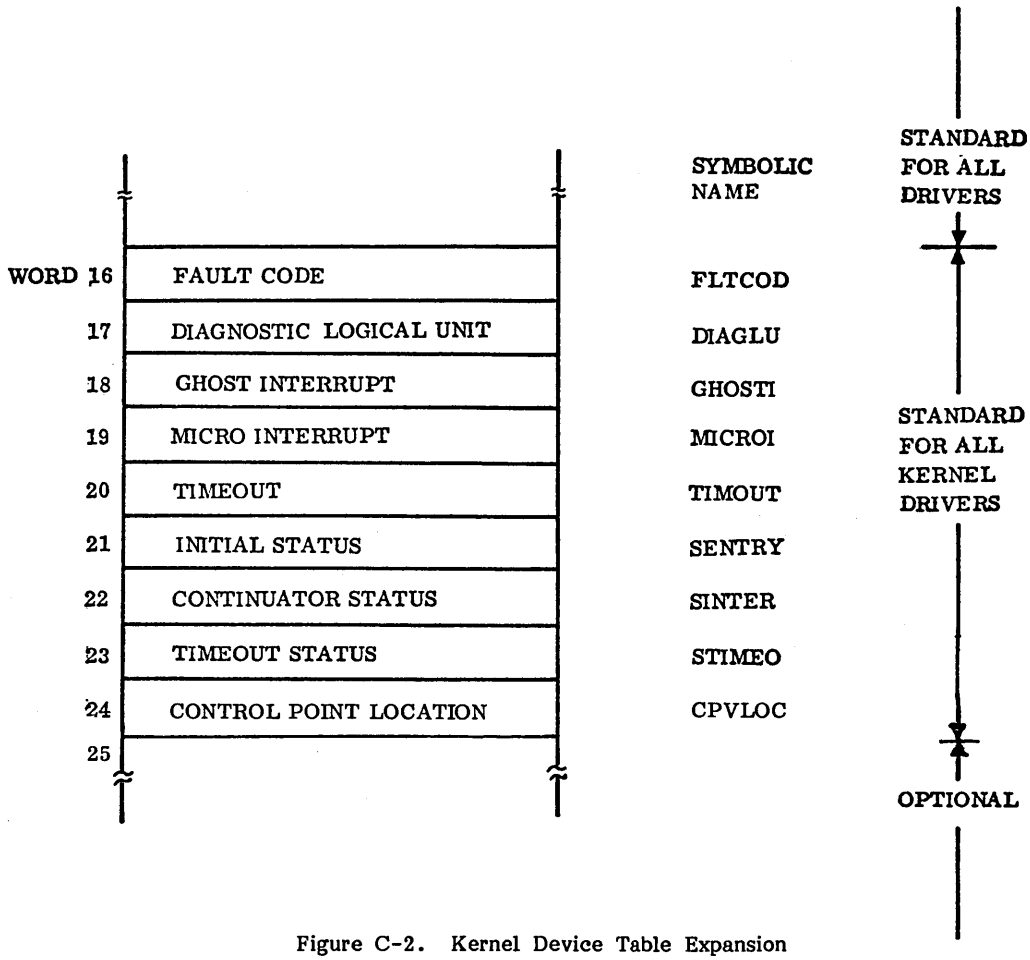


Figure C-2. Kernel Device Table Expansion

Word 23 STIMEO
The status after the device's interrupt has timed out.

Word 24 CPVLOC
Control point location. Used for DMA devices only.

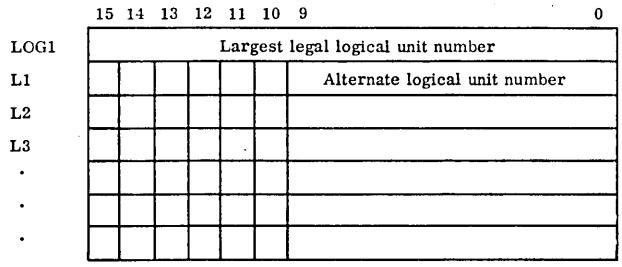
Word 25 and following These words may be added to the device table if required for special purposes. For example, they can be used to count the lines per page of output, link several tables together all using the same driver/kernel, or save multiple status words (the auxiliary status words should start at word 24).

LOGICAL UNIT TABLES

Three logical unit tables (LOG1, LOG1A, and LOG2) specify correspondences between logical and physical units, and between logical units and the threads of I/O tasks awaiting execution of those logical units.

LOG1 TABLE — ALTERNATE DEVICE TABLE

The logical unit table indicates whether a logical unit has an alternate physical device that can be used for the I/O transfer in the event that this physical device fails. Only one alternate unit can be used for a given logical unit. However, several logical units can in fact use one physical device.



Where: Bit 15 is reserved.

Bit 14 is 0 if the logical unit does not share the device.
1 if the logical unit shares a device with another logical unit.

Bit 13 is 0 if the logical unit is operative.
1 if the logical unit is out of service: the alternate, if any, is in use.

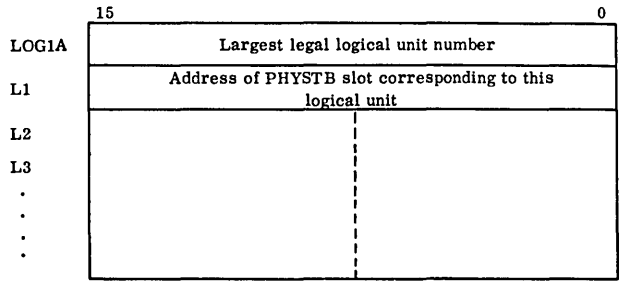
Bit 12 is flag bit for LU down message.

Bits 11-10 are reserved for future use.

Bits 9-0 are the alternate logical unit number.

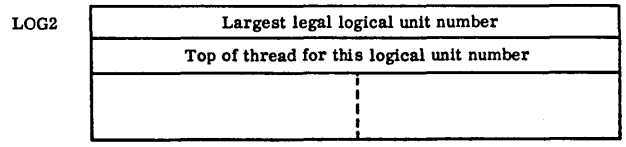
LOG1A TABLE — LOGICAL/PHYSICAL UNIT TABLE

This table relates each logical unit to its physical device table (PHYSTB). One physical device may have many logical units, hence many PHYSTBs, but each logical unit has one and only one PHYSTB.



LOG2 TABLE — TASK THREADS

The table contains a pointer to the first entry in each logical unit's thread. The threads may be extended throughout (protected) core. Entries in each thread are ordered on a FIFO within priority basis. Unprotected requests are not threaded to the I/O threads until execution time unless swapping is prohibited. At execution time, a copy of the request is saved in SYSDAT and this copy is threaded to the I/O device.



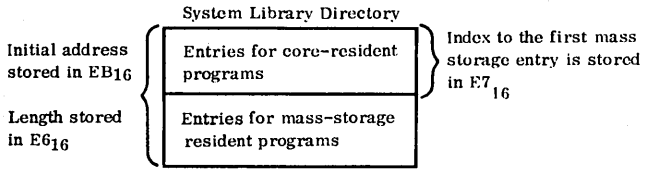
The two directories in the CYBER 18/1700 MSOS are used to locate system programs in core memory or mass storage and programs or files on mass storage.

SYSTEM LIBRARY DIRECTORY

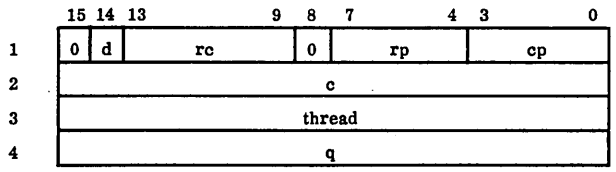
The system directory is core-resident and is divided into two parts.

- Entries relating to core-resident system programs (in core at all times)
- Entries relating to system programs that are resident on mass storage and placed in core only when needed

The system directory is constructed during system initialization. When it is complete, the system initializer stores its initial address in location EB_{16}. The index is the number added to the contents of EB_{16} to obtain the address of the first mass storage entry; it is stored in $E7_{16}$. The length of the system directory is stored in $E6_{16}$.



Entries for core-resident programs appear as follows:



Where: d is the part indicator
- 0 Indicates that a program runs in part 0
- 1 Indicates that a program runs in part 1

rc is the request code, which is set to 1 for core-resident programs.

rp has no meaning to the core-resident directory.

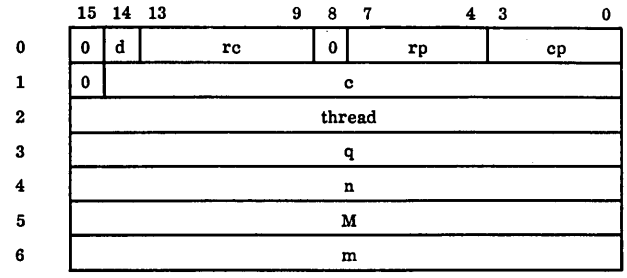
cp is the priority level at which the program is operated. It is set to the value of p in the parameter list of the program requesting the library program.

c is the core location of the initial execution address of the program.

thread is the location used to thread the entry into the scheduler thread.

q is the parameter to be passed to the program in the Q register when the program is entered; the contents of the Q register at the time the request was initiated.

Entries for mass-storage resident programs appear as follows.



Where: d is
- 0 Indicates that a program runs in part 0
- 1 Indicates that a program runs in part 1

rc is the request code. It is set to zero so that when the directory entry is threaded to the queue for the mass-storage logical unit, the driver will recognize the special form.

rp is the area of allocatable core available to this request. This is set by an *S command of LIBEDT. If d is 1, rp determines the priority in the partition thread.

cp is the priority level at which the program is operated. It is set to the value of p in the parameter list of the program requesting the library program.

c is the lowest (toward zero) memory address allocated for this request. It is determined by a SPACE request, after core in which the program can be operated has been allocated. If d is 1, c is fixed to the starting address of the partition into which this program was absolutized.

thread is the location used to thread the entry into the scheduler thread.

q is the parameter to be passed to the program in the Q register when the program is entered (the contents of the Q register at the time the request was initiated).

n is the program length in words.

M is always zero.

m is the mass storage address; it contains the starting sector address of where the program begins.

When the system initializer substitutes an index for an external name, it proceeds as follows:

Core-resident entry	(ordinal - 1) * 4
Mass-storage-resident entry	(ordinal - 1) * 7 + ($E7_{16}$)

The ordinal represents the position of the program among others of its type (core or mass storage) in the system directory. For example, both the fifth core-resident program and the fifth mass-storage-resident program would have the ordinal 5.

JOB PROCESSOR FILE DIRECTORY TABLE

Files defined and used by job processor routines (refer to section 9) are listed in the job processor file directory table which is available on mass storage. Each file entry in the table consists of nine words, with ten file entries per sector. The first sector LSB of the job processor file directory table is available in the extended core table ($E9_{16}$) + 8.

The total number of files used for the job processor file directory table is in SYSDAT as EQU JBFLV4 (the number of files).

The following is the format of a file entry:

	15 14	8 7	0	
0	Character 1		Character 2	File name (ASCII)
1	Character 3		Character 4	
2	Character 5		Character 6	
3	Character 1		Character 2	Security code (ASCII)
4	Character 3		Character 4	
5	Character 5		Character 6	
6	O/C	Character 1 (m)	Character 2 (m)	Expiration date (ASCII)
7	W/R	Character 3 (d)	Character 4 (d)	
8	Character 5 (y)		Character 5 (y)	

Where: file name is the file name, stored in ASCII format. If the name consists of less than six characters, blanks are used for the trailing characters.

security code is the security code, stored in ASCII format. If this code is of less than six characters, blanks are stored as trailing characters.

expiration date is the expiration date, stored in ASCII format. The six-character date is of mm dd yy format.

mm	Month – 01 through 12
dd	Day – 01 through 31
yy	Year – 00 through 99

O/C is the OPEN/CLOSE status of the file.

1	File is open.
0	File is closed.

W/R is the WRITE/READ status of the file.

1	File can be read or written.
0	File is read only.

PROGRAM LIBRARY DIRECTORY

The program directory is stored entirely on mass storage in a linked form: each link points to the next link and each link occupies a sector. The first sector in the linked list has its number stored in core locations $C3_{16}$ (MSB) and $C4_{16}$ (LSB).

The program library directory contains the ASCII names of all entry points in the program library together with their beginning mass storage addresses. The names of all permanent files are stored in the program library directory. No entry point or file name appears twice however, an entry point name and file name may be the same. Names are one to six characters; the first character must be alphabetic and the remaining characters are alphanumeric.

A representative sector in the program directory is shown below:

1	A	B	A six-character name
2	C	D	
3	E	F	
4	The file/program designator †		
5	The starting sector number of the file or program		
	Additional sector entries (up to 18 in a sector)		
91			
92	Not used		
93			
94	Number of empty locations, if last sector		Zero, if this is the last sector in list
95	MSB of next sector number		
96	LSB of next sector number		

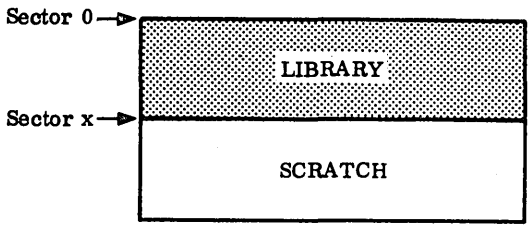
† If the entry is a permanent file, the fourth word contains the complement of the number of sectors that the file occupies; otherwise, it contains zero.

Disk sectors are numbered logically from 0 to n, where n is the number of sectors on the disk. Two consecutive words are used to specify a sector number. The 16 most significant bits (MSBs) of the sector address are contained in word 1 and the 15 least significant bits (LSBs) are contained in word 2. Bit 15 of word 2 is always zero. When a read or write requires more than one disk sector to satisfy the word count, consecutive sectors are accessed.

Since the system may access the disk between successive job requests, each user must specify the sector at which an operation begins. The system does not guarantee successive sectors on successive operations.

The library unit consists of two sections: library and scratch.

The library can be accessed by unprotected programs only through a GTFILE or LOADER request. Unprotected programs can access the scratch area directly by READ, WRITE, etc. The system can access the entire disk at any time by specifying the sector number. Unprotected programs cannot access the library portion of the library unit and need not be concerned with the beginning of the scratch area; they may address the scratch area logically from sector 1, etc. The library unit is arranged as follows.



When an unprotected program asks for sector 1 on the library unit, it is given sector x, which is computed by adding the sector defined in locations $C0_{16}$ and $C1_{16}$ to value 1. $C0_{16}$ and $C1_{16}$ define the end of library and the start of scratch for the library unit only. Units not containing the library have their scratch starting at physical sector 1.

The library and scratch units may be physically different.

On the library unit, sectors 0 through 4 are used for the system autoload area.

TABLE F-1. MSOS STANDARD LOGICAL UNIT ASSIGNMENT

Logical Unit	Function
1	Core allocator
2	Dummy
3	Dummy
4	Comment I/O
5	COSY driver – Unit 0†
6	Magnetic tape – Unit 0†
7	Pseudo tape or tape simulator – Unit 0†
8	Library unit
9	Standard list unit
10	Standard input unit
11	Standard output unit
12	FORTRAN list unit

†If not present in a system, dummy is substituted.

TABLE F-2. DRIVER PRIORITY LEVELS

Equipment	Level
601 Magnetic Tape (Unbuffered)	14
1721, 1723, 1777 Paper Tape I/O	14
1729-2, 1729-3, 1728-430 Card I/O	14
1829-30/60 Card Reader	14
1711, 1713, 711 CDT, 1811-1 Console Display	13
364-4 Communications Multiplexer	12
Remote 1500 equipment	12
1744 Digigraphics Controller	11
Dummy driver	10
601 Buffered Magnetic Tape	10
1832-5 Cassette Tape	10
608, 609, 615, 616 Magnetic Tape	10
1860-72/92 Magnetic Tape	10
1860-5/6 Magnetic Tape	10
1742-501, 1742-30/120 Printer	10
1746-405 Card Reader	10
1747 Data Set Controller	10
1743-2, 1595 Communications Multiplexer	10
1745-2 Buffered Display Controller	10
1733-1, 1733-2, 1738, 1739, 1833-1, 1751, 1752, 1833-4 Mass Memory	9
1833-5 Flexible Disk	9
1536 Relay Analog Multiplexer	9
COSY driver, extended memory driver	8
Pseudo tape, magnetic tape simulator	8
Core allocator	7

TABLE F-3. SOFTWARE PRIORITY ASSIGNMENTS

Level	Use
15	Stall/parity/protect violations
14	Card reader/punch, unbuffered magnetic tape
13	Timer, low-speed I/O
12	Communications multiplexer, remote 1500 equipment
11	Digigraphics
10	Line printer, message buffering, magnetic tape
9	Analog input, disk drum
8	DACS scan, direct digital control, in-line responses (AUTRAN); TIMESHARE Executive
7	Core allocation
6†	Special application programs (IMPORT)
5†	DACS alarm, AUTRAN supervisory, CRT console handler (AUTRAN)
4†	Calculations, logs (AUTRAN); Timeshare user programs; SCMM
3	Manual interrupt processing, magnetic tape rewind
2	Job cancel, idle if swapped
1	Job I/O completion
0	Job processing, AUTRAN and FORTRAN compilation
-1	Idle

†Re-entrant FORTRAN levels

TABLE F-4. STANDARD EQUIPMENT CODES AND INTERRUPT LINES FOR 1700 MSOS

Device Type	Interrupt Line And Equipment Code
Low-speed I/O line 1 device	1
Drum mass memory	2
1747 Data Set Interface	3
Line printers	4
Communications controller	5 and 6
Magnetic tape	7
1500 Series equipment†	8 and 9
Card punch	10
Card reader	11
1744 Digigraphics Controller	12 and 13
1745-2 Display Controller	12 and 13
Unassigned	14
1781-1 Floating Point Unit	15

†The 1590 also uses interrupt line 6. The 1595 also used interrupt line 5.

††The 1576 also uses interrupt line 15.

TABLE F-5. STANDARD EQUIPMENT/INTERRUPT ASSIGNMENTS FOR CYBER 18-20 AND 18-30

Peripheral	Equipment† Code	Macro Interrupt	Micro Interrupt
Teletypewriter/Console Display	1	1	1
Paper Tape Reader	2	2	2
Paper Tape Punch	2	2	2
Card Punch	2	2	2
None	3	3	3
Line Printer	4	4	4
None	5	5	5
None	6	6	6
Cassette	7	7	7

†Equipment codes 0, 3, 5, 6, and 8 are currently unassigned and reserved for future use.

TABLE F-5. STANDARD EQUIPMENT/INTERRUPT ASSIGNMENTS FOR CYBER 18-20 AND 18-30 (Contd)

Peripheral	Equipment† Code	Macro Interrupt	Micro Interrupt
Clock	1	8	8
Magnetic Tape Transport (NRZI only)††	9	9	0, 9
Eight-Channel Communications Line Adapter	10	10	10
Dual-Channel Communications Line Adapter	10	10	10
Card Reader	11	11	11
Magnetic Tape Transport (NRZI and Phase Encoded)	12	12	†††
IOM	13	13	†††
Storage Module Drive	14	14	†††
Cartridge Disk Drive	14	14	†††
Flexible Disk Drive	15	15	†††
Protect, Parity and Power Failure (internal)	†††	0	†††
Macro Stop and Panel (internal)	†††	†††	12-15

†Equipment codes 0, 3, 5, 6, and 8 are currently unassigned and reserved for future use.

††The LCTT (NRZI only) micro interrupt is wired to both micro interrupt zero and nine. The software has responsibility to select the desired one.

†††Not applicable

The 1963 American Standard Code for Information Interchange (ASCII) is used by the CYBER 18/1700 MSOS. ASCII code uses eight bits: bit 8, which is always zero, is omitted in the table below. Bits 1 through 4 contain the low-order four bits of code for the character in that row. Bits 5 through 7 contain the high-order three bits of the code for the character in that column. The code is given in ascending sequence in table G-1.

TABLE G-1. ASCII CONVERSION TABLE

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
NULL	000 0000	0	Null/idle
SOM	000 0001	1	Start of message
EOA	000 0010	2	End of address
EOM	000 0011	3	End of message
EOT	000 0100	4	End of transmission
WRU	000 0101	5	Who are you
RU	000 0110	6	Are you
BELL	000 0111	7	Audible signal
FE_0	000 1000	8	Format effector
HT/SK	000 1001	9	Horizontal tab skip (punched card)
LF	000 1010	A	Line feed
V_{TAB}	000 1011	B	Vertical tabulation
FF	000 1100	C	Form feed
CR	000 1101	D	Carriage return
SO	000 1110	E	Shift out
SI	000 1111	F	Shift in
DC_0	001 0000	10	Device control/data link escape
DC_1	001 0001	11	Device controls
DC_2	001 0010	12	
DC_3	001 0011	13	
DC_4 (STOP)	001 0100	14	Device control/stop
ERR	001 0101	15	Error
SYNC	001 0110	16	Synchronous idle
LEM	001 0111	17	Logical end of media
S_0	001 1000	18	Information separators
S_1	001 1001	19	
S_2	001 1010	1A	
S_3	001 1011	1B	
S_4	001 1100	1C	
S_5	001 1101	1D	
S_6	001 1110	1E	
S_7	001 1111	1F	

TABLE G-2. ASCII TO EBCDIC CONVERSION

8-Bit ASCII Codes	171x-1 Teletypewriter Array	171x-2 Teletypewriter Array	EBCDIC Character	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape	Tape Code EBCDIC
20_{16}	Space	Space	Space	No Punch	No Punch	20_8	40_{16}
21†	!	!	!	11-8-2	12-8-7	52	5A
22	"	"	"	8-7	8-7	17	7F
23†	#	#	#	12-8-7	8-3	77	7B
24	$	$	$	11-8-3	11-8-3	53	5B
25†	%	%	%	0-8-5	0-8-4	35	6C
26†	&	&	&	8-2	12	00(35)††	50
27†	'	'	'	8-4	8-5	14	7D
28†	(	(	(	0-8-4	12-8-5	34	4D
29†	)	)	)	12-8-4	11-8-5	74	5D
2A	*	*	*	11-8-4	11-8-4	54	5C
2B†	+	+	+	12	12-8-6	60	4E
2C	,	,	,	0-8-3	0-8-3	33	6B
2D	-	-	-	11	11	40	60
2E	.	.	.	12-8-3	12-8-3	73	4B
2F	/	/	/	0-1	0-1	21	61
30	0	0	0	0	0	12	F0
31	1	1	1	1	1	01	F1
32	2	2	2	2	2	02	F2
33	3	3	3	3	3	03	F3
34	4	4	4	4	4	04	F4
35	5	5	5	5	5	05	F5

†To operate in 026 punched card mode, ASCII 63 options are selected. To operate in 029 punched card mode, ASCII 68 options are selected. These options are assembly-time options for each driver affected.

††Since 173x magnetic tape controllers do not provide any code conversion, BCD code 00 is illegal and causes a noise record or BCD code 35 is substituted for the illegal 00 code to prevent tape errors.

On tape write operations, the ASCII codes 25_{16} (%) and 26_{16} (&) are written as BCD 38_8.

On tape read operations, the BCD code 35_8 is always translated to an ASCII 25_{16} (%).

TABLE G-2. ASCII TO EBCDIC CONVERSION (Contd)

8-Bit ASCII Codes	171x-1 Teletypewriter Array	171x-2 Teletypewriter Array	EBCDIC Character	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape	Tape Code EBCDIC
36	6	6	6	6	6	06	F6
37	7	7	7	7	7	07	F7
38	8	8	8	8	8	10	F8
39	9	9	9	9	9	11	F9
3A	:	:	:	8-5	8-2	15	7A
3B	;	;	;	11-8-6	11-8-6	56	5E
3C†	<	<		12-8-6	12-8-4	76	CE
3D†	=	=	=	8-3	8-6	13	7E
3E†	>	>		8-6	0-8-6	16	EC
3F†	?	?	?	12-8-2	0-8-7	72	6F
40			Rej.	0-8-7	8-4	37	3F
41	A	A	A	12-1	12-1	61	C1
42	B	B	B	12-2	12-2	62	C2
43	C	C	C	12-3	12-3	63	C3
44	D	D	D	12-4	12-4	64	C4
45	E	E	E	12-5	12-5	65	C5
46	F	F	F	12-6	12-6	66	C6
47	G	G	G	12-7	12-7	67	C7
48	H	H	H	12-8	12-8	70	C8
49	I	I	I	12-9	12-9	71	C9
4A	J	J	J	11-1	11-1	41	D1
4B	K	K	K	11-2	11-2	42	D2
4C	L	L	L	11-3	11-3	43	D3
4D	M	M	M	11-4	11-4	44	D4
4E	N	N	N	11-5	11-5	45	D5
4F	O	O	O	11-6	11-6	46	D6
50	P	P	P	11-7	11-7	47	D7
51	Q	Q	Q	11-8	11-8	50	D8
52	R	R	R	11-9	11-9	51	D9

†To operate in 026 punched card mode, ASCII 63 options are selected. To operate in 029 punched card mode, ASCII 68 options are selected. These options are assembly-time options for each driver affected.

TABLE G-2. ASCII TO EBCDIC CONVERSION (Contd)

8-Bit ASCII Codes	171x-1 Teletypewriter Array	171x-2 Teletypewriter Array	EBCDIC Character	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape	Tape Code EBCDIC
53_{16}	S	S	S	0-2	0-2	22_8	E2
54	T	T	T	0-3	0-3	23	E3
55	U	U	U	0-4	0-4	24	E4
56	V	V	V	0-5	0-5	25	E5
57	W	W	W	0-6	0-6	26	E6
58	X	X	X	0-7	0-7	27	E7
59	Y	Y	Y	0-8	0-8	30	E8
5A	Z	Z	Z	0-9	0-9	31	E9
5B†	[	[	\	12-8-5	12-8-5	75	4A
5C†	\	\	|	0-8-2	0-8-2	36	FA
5D†	]	]	⌡	11-8-5	11-8-5	55	CC
5E	↑	^	--	11-8-7	11-8-7	57	6D
5F†	←	-	¬	0-8-6	0-8-5	32	5F
60	\	\	\			N/A	79
7B	{	{	{			N/A	C0
7D	}	}	}			N/A	D0
7E	~	~	~			N/A	A1

†To operate in 026 punched card mode, ASCII 63 options are selected. To operate in 029 punched card mode, ASCII 68 options are selected. These options are assembly-time options for each driver affected.

The CYBER 18/1700 Mass Storage Operating System is designed to meet a large variety of user needs. Detailed descriptions of the customization processes can be found in the MSOS Customization Manual.

The system consists of a collection of standard software modules and a program containing all system variables and tables (SYSDAT).

Customization is performed by including or deleting various standard modules and by modifying SYSDAT. In addition, special user programs and data may be easily included in the system.

Installation of this software is performed by the system initializer routines, which load and link all system software in main memory and mass storage, based on initializer control statements that specify the installation.

Although the initializer is executed off-line (i.e., not as a part of MSOS), a certain amount of customization may be performed on-line by the use of the library editing feature. This includes such items as the modification of system library and program library programs.

CUSTOMIZATION

Customization of MSOS is performed by modifying SYSDAT. All standard MSOS modules make use of data in SYSDAT for configuration dependent information. This section provides a summary of the customizable data found in SYSDAT. The MSOS Customization Manual should be consulted prior to attempting any modifications.

COMMUNICATIONS REGION

An area in low core has been reserved for applications use. It is 108 words in length (47_{16} through $B2_{16}$) and is especially valuable because all locations in this region may be referenced directly with one word instructions.

INTERRUPT REGION

The area between 100_{16} and $13F_{16}$ is reserved for interrupt response. Each interrupt line is related by hardware design to a four-word block of memory in this area.

EXTENDED COMMUNICATIONS REGION

A table of frequently used MSOS parameters is referenced through location $E9_{16}$. Of particular interest are words 10 and 11, which indicate the memory bank in which unprotected core resides, and words 21 and 22, which specify the maximum sector of scratch.

SYSTEM IDENTIFICATION

An 18-word table that contains an alphanumeric system identification and the date the system was installed is located immediately following the extended core table. This information is printed on the commnet device each time an autoload is performed. The installation date is specified by *S control statements in the installation file. This table is useful in identification of the system from a core dump.

STORAGE STACKS

The storage stacks are divided into two customizable parts. The size of volatile storage is dependent upon the configuration and is based on the number of re-entrant FORTRAN priority levels, as well as additional user requirements. The scheduler/timer stack holds requests unitl the system priority level conditions dictate their execution. This stack should be sized to contain all expected simultaneous requests.

LOGICAL UNIT TABLES

Each logical unit in the system must have an entry in the LOG1A, LOG1, and LOG2 tables and may require an entry in the Diagnostic Timer (DGNTBL) table. Refer to appendix C for a description of these tables.

STANDARD LOGICAL UNITS

Equivalences defining the standard dummy, comment I/O, library, scratch, list, input, and binary output units are contained in SYSDAT.

LINE 1 TABLE

Some configurations require several logical units that are connected to the line one (low-speed I/O) interrupt. In this case a list of physical device table addresses is required by the line one handler routine.

PHYSICAL DEVICE TABLES

Each logical unit in the system generally requires a physical device table, which contains information necessary to the device's driver. Refer to appendix C for a description of these tables.

CORE-RESIDENT DATA

Several items of data are required to specify the manner in which core memory is arranged. BGNCOR specifies the first word address of the core-resident monitor, and this

address must be changed if additional modules are included in this part of core. The core area size parameters (N5 through N15) allow specification of allocatable core areas. The partition core tables are used to define the address of each core partition. The system common declaration defines the amount of blank common in the system.

SYSTEM IDLE LOOP

This program is executed when there are no other tasks to perform in the system. In addition, a section of this routine is executed the first time the idle loop is entered following an autoload, and may be used to perform user start-up functions.

A counter of idle loop cycles is maintained in the location labeled IDLCTR. This may be used by an applications program to calculate system idle time.

SYSTEM TIMER INTERRUPT RESPONSE

The 1700 MSOS supports six different types of timing devices. The interrupt response for the particular configuration is contained in SYSDAT. After acknowledging and reactivating the timer interrupt, the timer response routine is executed to allow completion of timer requests and the calculation of the time of day. The CYBER 18 has one timer type with the timer tables and interrupt response in SYSDAT.

BUFFERED DATA CHANNEL TABLES

Data tables are required by the 1706 handler if any system hardware controller share a buffered data channel.

A/Q ALLOCATION TABLES

Certain system devices require allocation of the A/Q channel during input/output operations in order to avoid lost data. The A/Q channel allocator routine uses these tables for this purpose.

MASS-RESIDENT DRIVERS BUFFER

This buffer is sized to hole mass-resident drivers during execution. It must be of sufficient size to contain the largest mass-resident driver in the system. It may, however, be sized to contain any two drivers simultaneously, or to contain some drivers simultaneously and some singularly.

RE-ENTRANT FORTRAN MASK

This location (FMASK) contains a bit that is equal to 1 for every re-entrant FORTRAN priority level in the system.

SYSTEM TIMER PARAMETERS

This section contains parameters related to the system timer, including the specification of the expected interrupt frequency.

SYSTEM OVERLAY SIZES

Certain system library programs are designed to operate as overlays. This table specifies the size of the first overlay for each of these programs.

IOM PARAMETERS

This section contains information defining the physical configuration of any 1500 Series devices in the system, included are station assignments, the digital output history table, and digital input/output block assignment tables.

SYSTEM CHECKOUT PARAMETER

The starting sector of the failed core image to be used by the system checkout bootstrap is contained here. This image must be contained on the library unit and must be large enough to hold the physical size of core memory in the system.

FILE MANAGER TABLES

There are three items contained in this section. General parameters required by the file manager, the logical unit tables for each mass storage device that contains file information, and data required for the background job files. Refer to the File Manager Reference Manual for a detailed description of the file manager tables. A description of job files is contained in section 9 of this manual.

SYSTEM PRESETS

System presets allow unprotected reference to the foreground, which would otherwise result in protect violations. Care should be exercised in adding items to this table, as this tends to degrade the protected system.

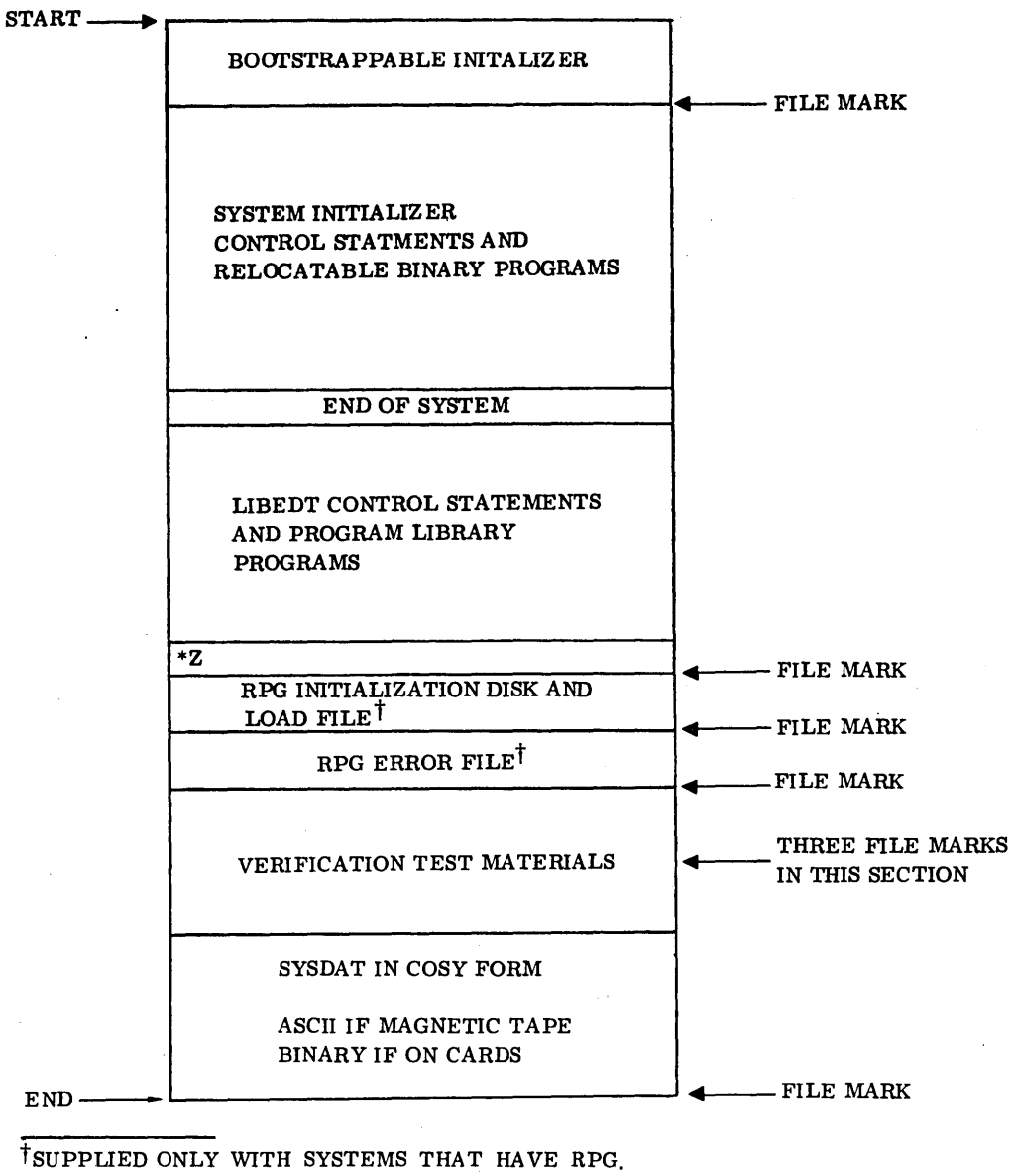
INSTALLATION

Installation of CYBER 18/1700 MSOS is performed by the use of the system initializer routines, which accept the installation file containing initializer control statements and relocatable binary programs.

Under MSOS, an attempt has been made to allow system installation to be performed in an extremely simple and straightforward manner. This section contains an overview of the procedures necessary to install CYBER 18/1700 MSOS. The system installation manual should be consulted for detailed information.

The MSOS user receives an installation file that has been configured, built, and tested to the required configuration. This installation file may be contained on seven- or nine-track magnetic tape, punched cards or paper tape, and is divided into sections as shown in figure H-1.

In order to install the system, the user has two options. If an operating MSOS system is available, the system initializer may be loaded from the program library and executed. In this case, the installation file should be advanced past the



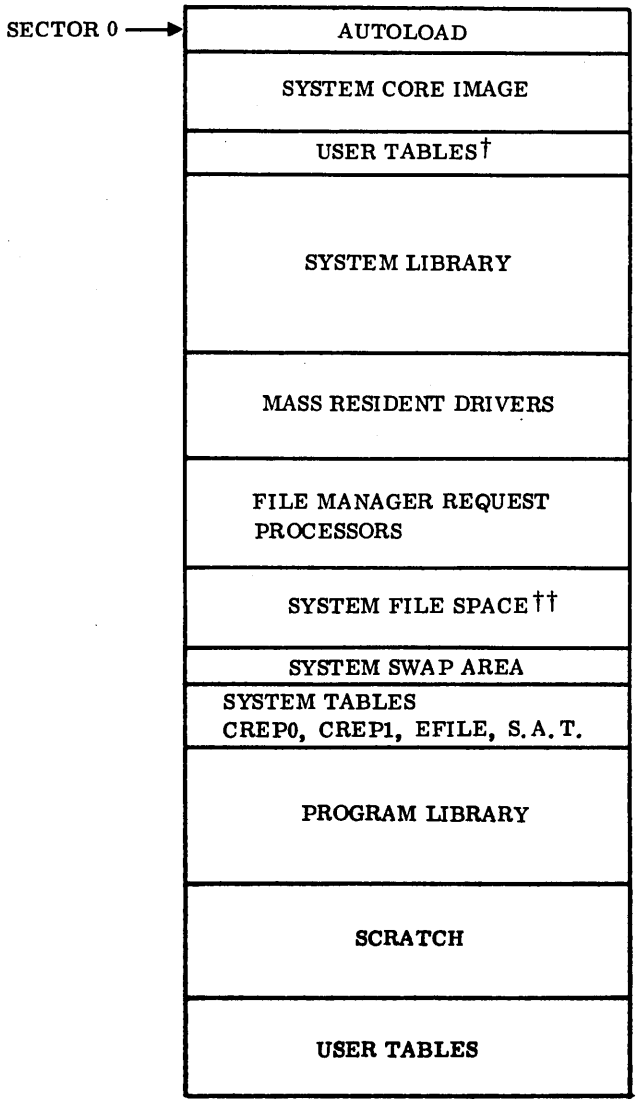
†SUPPLIED ONLY WITH SYSTEMS THAT HAVE RPG.

Figure H-1. MSOS Installation File

first file mark prior to installation. If a system is not available, a short hand loaded bootstrap must be entered and executed. On the CYBER 18 with the deadstart feature, a deadstart card deck to read seven- or nine-track tape is supplied. This will load the system initializer and allow it to be executed.

The system initializer processes the installation file and builds the core and mass resident portions of the system until the *T control record is encountered. At this time, a request to autoload the system is output on the comment device. Following autoload, the user initiates the library edit routine by entering *BATCH after a manual interrupt. LIBEDT then processes the next portion of the installation file. This allows the program library to be loaded and is terminated by an *U control statement. The system is now completely installed and may be verified by initiating the MSOS verification tests. All material required to execute these tests is contained in the next portion of the installation file. Once the tests are successfully completed, the system may be put into operation or customized as the user desires.

The installation of MSOS 5 into core and mass memory is shown in figures H-2 and H-3.



† OPTIONAL ITEMS.
†† FOR FIXED PLATTER DISK, THIS AREA IS MOVED TO THE FIXED PLATTER.

Figure H-2. Mass Memory Arrangement (Library Unit)

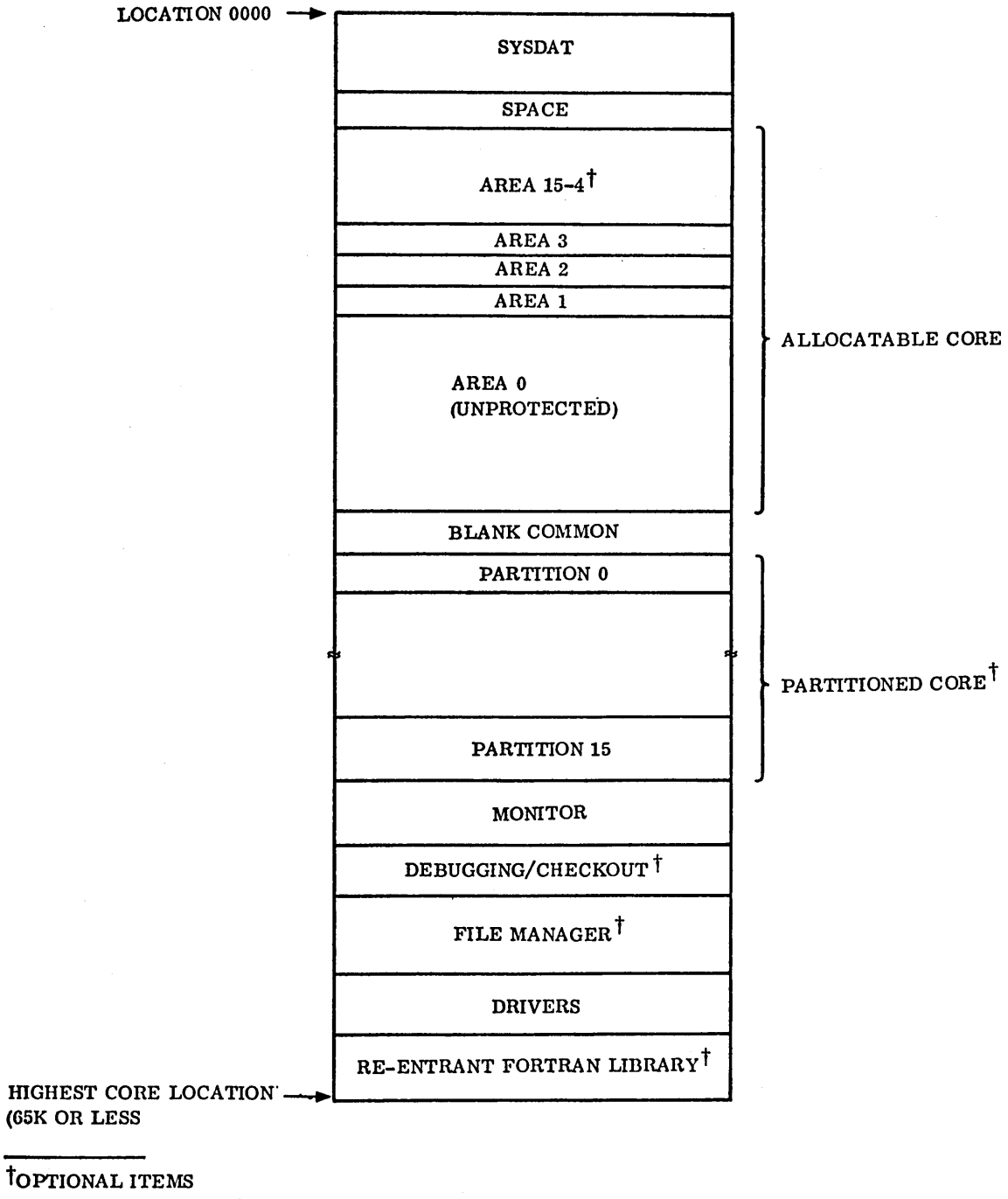


Figure H-3. Memory Arrangement

BACKGROUND PROGRAMMING OVERVIEW I

Available to the CYBER 18/1700 MSOS user is the capability to execute programs in batch mode, unprotected core, and at low priority. Use of unprotected core provides for system protection from undebugged programs, while execution at a low priority ensures that higher priority real-time functions may be completed as required. The job processor is the supervisory program for unprotected programs.

The following capabilities are available to the background programmer:

- System requests as defined in section 3 – The job processor validates unprotected requests prior to queueing them to protect the system from undebugged programs.
- Protected core communications region (refer to appendix B) containing fixed constants for operand masking – In addition, any protected core location may be read only.
- Protected core communication capability by use of a table of preset entry points (refer to Unprotected Entry Points, section 2) – This allows unprotected programs to utilize protected core subroutines.
- Relocatable binary loader for program loading and subroutine linking – Common subroutines may be stored on the mass storage program library and called by unprotected programs. At load time these are loaded with the main program.
- Job files for assignment as data storage areas for simulation of magnetic tape input/output on disk (refer to Mass Storage Job File Handling Statements, section 9).
- File manager for storage and management of data in permanent files – The background user may access protected files and prepare reports or create, maintain, and analyze his own unprotected files (refer to section 5).
- On-line debug, on-line trace, breakpoint program, and recovery program debugging aids to enhance program checkout (refer to section 10).
- Library editing program (refer to section 13) for program and system library maintenance.
- Editor program to create and modify files (refer to section 14).

DEVICE FAILURE CODES

J

TABLE J-1. DEVICE FAILURE CODES

Device Failure Code	Error	Significance
0	Time-out	Failure to interrupt within allotted time (requires TIMER package) Teletypewriter: Operator failed to supply input within allotted time. Ignore message and continue normally. All Other Devices: Hardware failed to generate an interrupt within the allotted time. Hardware maintenance required.
1	Lost data	Data not transferred out of read register before the next data word appeared. 1711/1713 Teletypewriter: Retype statement. 1829-30/60 Card Reader (diagnostic logical unit only); bad initiator status. 1833-5 Flexible Disk: Bad initiator pseudo status Magnetic Tape: Use CU option to continue without processing lost record or abort the read option.
2	Alarm	Indicates the presence of an abnormal condition 1713 Paper Tape Reader: Paper tape motion failure. No change in feed hole circuit has occurred for 40 milliseconds while trying to read. If not end-of-tape, manually position the paper tape so that the end of the next to last record and the beginning of the last record are on opposite sides of the photocells. If end-of-tape, take the CU option. Paper Tape Punch: Paper tape supply low or tape break. Abort punch operation and correct the problem. Line Printer: Paper out, paper tear, fuse alarm, or interlock open. Correct problem and use the RP option. 1729-2 Card Reader: Interlock open. Correct the problem and take the RP option. 1728/430 Card Reader: Interlock open or chip box full. Correct the problem and take the RP option. 1726/405 Card Reader: If output stacker is full, clear the output stacker and type RP. If card jam has occurred, abort the operation and correct the problem. If there is a failure to feed, attempt to ready the device and take the RP option. 1829-30/60 Card Reader: (diagnostic logical unit only: Bad continuator status 1832-5 Cassette Tape: Runaway tape 1833-5 Flexible Disk Drive: Bad continuator pseudo status COSY driver: First record is not a CSY/ control record.

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
2	Alarm (contd)	Magnetic tape simulator: Failure to fulfill request due to mass storage device error or illegal parameter in FILMGR request. Pseudo tape: Failure to fulfill request due to mass storage device failure or illegal parameter in FILMGR request.
3	Parity error	1711/1713 Teletypewriter: Attempt recovery by retyping the command. 1713 Paper tape reader: Manually position the paper tape so that the end of the next to last record and the beginning of the last record are on opposite sides of the photocells. Repeat the read request by typing RP in response to the error message. Magnetic tape: Tape is positioned after the bad record. Either tape the CU option to continue processing (the bad record will be ignored) or abort the operation. COSY driver: Last record was not an END/ record. 1833-4 Cartridge Disk: DMA parity error
4	Checksum error	(FREAD binary) Sum of the header word and data in a record did not balance to zero when added to the checksum word. Paper tape reader (1713): Plugged feed holes or dirty paper tape. Check the tape for dirt and plugged feed holes. Attempt recovery by manually positioning the paper tape so that the end of the next ot last record and the beginining of the last record are on opposite sides of the photocells. Repeat the read request by typing RP in response to the error message. Card readers: Holes are not cleanly punched. Check cards for tears between holes. If cards are all right, attempt recovery. Otherwise, perform the following operations: 1. Remove cards from the input hopper 2. (1729-430/1729-2/1729-3 only) Single cycle the card in the transport area to the output stacker. 3. Take the last two cards in the output hopper and put them into the input hopper ahead of the unread cards; with a multicard record, re-read all cards within the record. 4. (1726-405 only) Press the RELOAD memory switch. 5. Ready the card reader. 6. Take the RP option. 1833-5 Flexible Disk Drive: Status faults after I/O. COSY Driver: No end-of-file mark following the END/ record. I/O device did not send reply to the computer within the allotted time.
5	Internal reject	The computer cannot communicate with the device. Check the hardware address switch and POWER ON switch. The RP option may be used if the problem has been corrected. COSY driver: Read on the write unit or write on read before the end-of-deck marker was encountered.

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
6	External reject	The I/O device has replied to the computer that it is not ready to perform specified request. The device is busy or not ready. If the device is not busy, check the ready switch. Attempt to continue by typing RP. COSY driver: Motion request is on read unit after CSY/record and before end-of-deck marker.
7	Compare	Hardware problem - A compare error occurs when a faulty signal is detected in the area of the punch solenoid and echo amplifier circuits during an echo check. 1728-430 Card Reader: Remove and discard last card punched. Ready device and type RP. Card readers: Attempt recovery as for card checksum error (see code 4).
	Pre-read	A pre-read error occurs if all read amplifiers are not off during a dark check. 1728-430 Card Reader: Remove and disacrd the last card punched. Ready the device and type RP. Card readers: Attempt recovery as for card checksum error (see error code 4).
8	Illegal Hollerith punch	Occurs when the card reader encounters a punch sequence that does not comply with the Hollerith to ASCII conversion table being used by the driver. To allow software recovery, the driver places an ASCII ? in the buffer word for the bad column. Select the repeat option to continue, or abort the job and correct the mispunched cards.
9	Sequence	Cards within a record are not in sequential order. Abort the read operation and restore sequential order to the record.
10	Non-negative record length	The first word of a formatted binary record is the complement of the number of records within the record. The word may be a negative number indicating that the card read was not the first card of the record. Attempt recovery using the procedure for checksum error (see error code 4).
11	Read/write mode change	Indicates a switch from read or write mode 1728-430 Card Reader: This message is issued only as a warning to the operator. If mode switch is allowable, repeat the request using the RP option.
12	7/9 punch	The error occurs if a 7/9 punch in column 1 is read when an FREAD ASCII request is specified.

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
12	7/9 punch (contd)	Card reader recovery: 1. If column 1 is a 7/9 punch, no recovery; abort operation request is wrong mode. 2. If column 1 was misread, read card as for checksum error. Magnetic tape: No write ring installed.
13	Can't write on device	Attempt was made to write on magnetic tape without write enabled. Insert write ring and use RP option. Pseudo tape: Attempt to write on file that was opened to read only Magnetic tape simulator: Attempted write with write ring not enabled. See manual input operations. 1832-5 Cassette Tape: Write not enabled 1833-5 Flexible Disk: Write enable switch not set or diskette has been defined as read only via motion request (code = 5). 1833-4 Cartridge Disk: Write not enabled.
14	Not ready	Ready the device and use the RP option.
15	Noise record	1832-4 Magnetic Tape: A noise record was detected and ignored.
16	Controller seek	The controller seek error occurs when the controller has failed to obtain the file address selected during a read, write, compare, or checkword operation. This is usually an indication of a positioning error.
17	Drive seek	The drive seek error occurs when the drive unit detects that the cylinder positioner moved beyond the legal limits of the device during a load address, write, read, compare, checkword, check, or write address function.
18	Address	This error occurs when an illegal file address obtained from the computer is detected, or the controller has advanced beyond the limits of file storage. Magnetic tape simulator: Attempted read past end of written data. 1833-5 Flexible Disk: Requested sector area for I/O does not fit within the 75 logical tracks that can be addressed or initialization of diskette attempts to reference track beyond the valid 0-76. 1833-4 Cartridge Disk: End of medium
19	Protect	The protect fault occurs when an unprotected controller operation attempts to write in a protected core location. 1833-4 Cartridge Disk: DMA protect fault
20	Checkword	The checkword error occurs when the controller logic detects an incorrect checkword in data read from file storage during a read, compare, or checkword operation.

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
20	Checkword (contd)	1833-5 Flexible Disk: Data written to diskette is not the same as data read from diskette when the software compare option selected via motion request (code = 3).
21	End-of-tape error	1832-5 Cassette Tape: End-of-tape is an unrecoverable error; tape automatically rewinds on next back motion command.
22	Card output stacker full	1728-430/1729-2/1729-3/1829-30/60 Card Readers: Empty output hopper and take the RP option.
23	Card input hopper empty	If the read operation is complete, use the CU option; otherwise, supply more cards and take the RP option.
24	Card feed	The read ready station does not contain a card after a feed cycle has occurred, and the input hopper is not empty. 1728-430/1729-2/1729-3/1829-30/60 Card Readers: Card feed failure error can occur as a result of warped or damaged cards. If the card reader can be made ready, take the RP option.
25	Card jam	A card transport problem has occurred. It is possible for a card jam to occur in any one or more of four read stations in the 1728 Card Reader. CAUTION Do not attempt to single-cycle the machine. Damage to the card transport or punch head may result. Call Customer Engineering to aid in clearing the jam. Jam While Reading 1. Examine the transport area. 2. Remove all cards that have completely passed under the read station. 3. The cards that have not completely passed the read station have not been read. Put these cards back into the hopper. Ready the card reader and repeat the request via the RP option. The cards must be recycled in proper sequence. 4. If the procedure results in failure, abort the read. Jam While Punching 1. Clear the jam 2. If a card has only partially passed the punch station, it has not been punched correctly. Discard the card. 3. Ready the card reader and type RP. If any cards were damaged, the operation may have to be started over to obtain a readable deck. 1829-30/60 Card Reader: Stacker jam status returned.

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
26	Insufficient file space	Not enough file space available for this request to the pseudo tape driver.
27	Device message	Illegal device message on 1720 punch
28	File	No file assigned to this logical unit (pseudo tape driver)
29	Read	A read error occurred in reading resident mass-storage driver.
30	Validation error	The frame punched does not compare with the original data, or echo error on 1720-1 punch. Abort the punch operation.
31	Short record	Attempt to write a record with a length less than the standard noise record length. Magnetic tape simulator: Noise record. Attempt to do zero length write.
32	Tape error	1720-1 Punch: Tape supply low 1860-5/6 Magnetic Tape: Tape error
33	Line break	Line break occurred while attempting to input on the 361-1 Communications Adapter.
34	Data interrupt	Data interrupt occurred after reading 80 columns. 1728-430/1729-2/1729-3/1829-30/60 Card Readers: This error indicates a hardware failure, possibly due to improper card travel; reread the card (see the recovery procedure for error code 4).
35	End-of-operation	End-of-operation interrupt occurred prior to reading 80 columns. 1728-430/1729-2/1729-3/1829-30/60 Card Readers: Continuous failures may indicate card slippage in feeding - Reread the card as for error code 4.
36	TX parity error	1860-5/6 Magnetic Tape: Transmission parity error
37	Wrong address	Buffered data channel is using the first word address other than the address sent by a buffered driver.
38	Not used	
39	Not used	
40	Repeated the request due to an error	The driver is attempting recovery.
41	Incomplete request	The request was not successfully completed. The driver attempted to repeat the request the maximum number of times.
42	Timing error	Occurred while drum was busy
43	Incomplete directory call or overlay read request	Due to irrecoverable error
44	Guarded address	Error on write Magnetic tape simulator: Attempt to write past the end of the specified magnetic tape simulator disk area.

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
45	Timing	Occurred while drum was not busy
46	External reject	On output
47	External reject	On input
48	Controller address	Controller address status not the expected value
49	Drive address	Drive address status not the expected value
50	No ID	1732-2, 1732-3, 1860-5/6: ID abort, no ID burst
		1833-4 Cartridge Disk: Missing index sector pulse
51	Illegal density	Attempt to select illegal density (1732-2, 1732-3) or attempt to select density when unit not at load point.
52	Power failure	Power failure on 1752 Drum
53	EOP	EOP not set after interrupt (1752 Drum)
		1829-30/60 Card Reader: No end-of-operation status.
54	Data	Data not set after interrupt (1752 Drum)
		1829-30/60 Card Reader: No data before end-of-operation.
55	Status	Bad status (an indeterminate error occurred on 1752)
56	Mass memory buffer expired	No more buffer space available (software buffer driver)
57	Buffer transfer	Mass memory error on buffer transfer, which is detected in the software buffer driver.
58	Not used	
59	PE lost data	Error in the phase encode formatter that affected the data transfer.
60	Illegal tape motion request	An illegal tape motion request was made to the magnetic tape simulator.
61	Interrupt status bit	1833-5 Flexible Disk and 1860-5/6 Magnetic Tape: Interrupt should not be set when initial status taken.
		1829-30/60 Card Reader: No interrupt status indication.
		1833-5 Flexible Disk: No interrupt status.
62	ADT	1829-30/60 Card Reader: Auto-data transfer indication fault status
63	Busy after EOP	1829-30/60 Card Reader: Still busy after end-of-operation occurs.
64	Not busy	1829-30/60 Card Reader: Not busy before end-of-operation occurs.
65	No interrupt selected	1833-5/1865-1 Flexible Disk: No interrupt select status bit when interrupt occurred.
66	Memory address	1833-5/1865-1 Flexible Disk: DMA memory address fault or A/Q transfer attempted to cross a bank boundary or DMA attempted to cross a bank boundary without priming request (motion request of code = 1).

TABLE J-1. DEVICE FAILURE CODES (Contd)

Device Failure Code	Error	Significance
67	Not used	
68	Interrupt status bit	1833-5/1865-1 Flexible Disk: Interrupt status bit not set when interrupt occurred
69	Initialization not enabled	1833-5/1865-1 Flexible Disk: Disk initialization switch not set
70	Connect	1833-1 Disk: Failure to connect to the control unit or drive after maximum retries.
71	ECC	1833-1 Disk: Error correction code could not correct error since too many error bits were generated.
72	Ghost interrupt	1833-1 Disk: Unexpected interrupt received
73	Force release	1833-1 Disk: Force release requested but disk not released (multiple disk adapter system.)
74	Transfer length	1833-1 Disk: Data transfer longer than requested
75	Transfer	1833-1 Disk: Data transfer was unaccomplished after maximum number of retries.
76	Recovery indicator	1833-5/1865-1 Flexible Disk: Informative error logged in the engineering file to indicate recovery has been performed on this device a specific number of times. Threshold value for error is contained in word 43 of the physical device table for this unit.
77	Expected reject did not occur	1833-5/1865-1 Flexible Disk: (diagnostic logical unit only) Illegal function issued but did not cause reject.
78	Transfer	1833-5/1865-1 Flexible Disk: The number of words transferred not correct or the spindle speed during initialization of disk more than 3.5 percent of normal value.
79	Unit busy	1833-5/1865-1 Flexible Disk: Unit busy at time I/O request is attempted.
80	Unit seeking	1833-5/1865-1 Flexible Disk: Unit seeking when I/O request is attempted
81	Unit doing I/O	1833-5/1865-1 Flexible Disk: unit doing I/O when I/O request is attempted
82	CU	1833-1 Disk: Error in 1833-3 control unit
83	Main memory address	1833-1 Disk: Disk adapter attempted to address a non-existent CPU memory address.
84	Bus relinquished	1833-4 Disk: Bus relinquished
85	CWA status error	1860-5/6 Magnetic Tape: Check word address status error
86	Switch mode error	1860-5/6 Tape: Attempt to read in binary mode a seven-track tape recorded in BCD or vice versa.
87	No character read in 25 feet	1860-5/6 Tape: No data found
88	DMA address error	1833-4 Cartridge Disk error

TABLE K-1. SYSTEM INITIALIZER CODES

Message	Significance	Message	Significance
ERROR 1	Asterisk initiator missing	ERROR 14	Data declared during an *M load but not by the first segment; initialization restarted
ERROR 2	Number appears in the name field		
ERROR 3	Illegal control statement	ERROR 15	Not used
ERROR 4	Input mode illegal	ERROR 16	Irrecoverable mass storage I/O error
ERROR 5	Statement other than *Y or *YM previously entered	ERROR 17	Irrecoverable loader error; last program loaded was ignored.
ERROR 6	Statement other than *Y previously entered	ERROR 18	Not used
		ERROR 19	Not used
ERROR 7	*Y not entered prior to the first *L	ERROR 20	*S, END0V4, hhhh not defined before first *L
ERROR 8	Name appears in the number field		
ERROR 9	Illegal hexadecimal core relocation field	ERROR 21	*S, MSIZV4, hhhh not defined before first *LP or *MP
ERROR A	Illegal mass storage sector number	ERROR 22	Attempt to load part 1 core resident into non-existent memory
ERROR B	Error return from the loader module		
ERROR C	Not used	ERROR 23	The name used in the second field of an *M control statement was not previously defined as an entry point.
ERROR D	Not used		
ERROR E	Field terminator invalid	ERROR 24	The entry point, SECTOR, was not defined at the start of initialization and is not available to the initializer.
ERROR F	More than 120 characters in the control statement	ERROR 25	Illegal partition number in the first field of an *MP statement or illegal number of partitions in the second field of statement
ERROR 10	Ordinal name without ordinal number		
ERROR 11	Doubly defined entry point	ERROR 26	An attempt was made to load an *MP program when no partitioned core table exists in SYSDAT.
ERROR 12	Illegal value		
ERROR 13	Loader control statement out of order – Correct order is L, LP, M, MP		

TABLE K-2. SYSTEM INITIALIZER LOADER ERRORS

Error	Significance	Message	Significance
LOADER ERROR 1	Unrecognizable input	LOADER ERROR 10	Unpatched externals
LOADER ERROR 2	Mass storage overflow	LOADER ERROR 11	Insufficient core for both SYSDAT and paging
LOADER ERROR 3	Out-of-order input block	LOADER ERROR 12	Illegal page number used
LOADER ERROR 4	Illegal data or common declaration	LOADER ERROR 13	Undefined transfer address
LOADER ERROR 5	Core overflow	LOADER ERROR 14	Invalid function for loader
LOADER ERROR 6	Overflow of entry point table	LOADER ERROR 15	Link table overflow
LOADER ERROR 7	Data block overflow	LOADER ERROR 16	External table overflow
LOADER ERROR 8	Duplicate entry point	LOADER ERROR 17	Entry point absolutized to $7FFF
LOADER ERROR 9	15/16-bit arithmetic error		

TABLE K-3. SYSTEM INITIALIZER DISK ERRORS

Error	Significance	Error	Significance
DISK ERROR	Address tag write sequence attempted but internal/external reject found	DISK COMPARE ERROR SECT aaaa WORD bbbb IS cccc SB dddd	Surface test pattern error on sector aaaa at word bbbb. Only one error will be listed per sector. Data read was cccc but it should be dddd.
DISK FAILURE xx	Surface test operation caused error xx. Refer to the device error codes to interpret xx.		

TABLE K-4. GENERAL SYSTEM ERROR MESSAGES

Message	Significance	Message	Significance
B01, statement	Statement or parameters are unintelligible for the breakpoint program.	L, nn FAILED xx ACTION	The number of the failed device appears when a driver cannot recover from an error Where: nn Logical unit of the failed device xx Code indicates the cause of failure
B02, hhhh	The specified hexadecimal address hhhh cannot be processed by the breakpoint program because it is protected.		
B03, hhhh	The breakpoint limit is exceeded. The specified hexadecimal address is the last breakpoint processed.	L, nn FAILED xx (yyyy) ACTION	Status informs the operator of device failure in the initializer. Where: nn Logical unit of the failed device xx Code indicates the cause of failure. (See appendix J for the code description.) (yyyy) The test hardware status of the failed device
CHECKING FILES – ERRORS	Errors detected in the file manager files check after autoload.		
DATE/TIME ENTRY ERROR	Re-enter MSOS date/time.		
DB FORMAT INCORRECT	Some part of remaining portion of request is incorrect for ODEBUG.		
DB INVALID REQUEST	Mnemonic does not agree with known mnemonic for ODEBUG.		
DB I/O ERROR	Monitor request return with error bit set for ODEBUG	LU nn DOWN	If a device is marked down, yet requested by a program, and this device contains no alternate, this message is typed on the comment device the first time it is requested after being downed. The completion address is always scheduled with error. The requesting program should not continually request downed units.
DB LHO/ LHC ERROR	Data written on mass storage does not match LHO/LHC input for ODEBUG		
DB NO CORE AVAILABLE	No allocatable core available for ODEBUG		
DB ORDINAL LENGTH ZERO	No program loaded in ordinal		
EF STACK OVERFLOW	Currently there is no space in the engineering file stack to record this device failure.	MI INPUT ERROR	Statement presented to the manual interrupt processor is unrecognizable or the requested program is not supplied.
EFSTOR LU ERROR	An attempt was made to update the engineering file for a logical unit less than 1 or greater than 99.	MM ERR xx LU = nn T = hhmm:ss S= ssss	Mass storage input/output error Where: xx Error number nn Logical unit hhmm Hours/minutes ssss Hardware status
EFSTOR MASS MEMORY ERROR	An error occurred in updating the engineering file on mass memory.		
ILLEGAL PARAMETERS SPECIFIED	Disk-to-tape has detected a non-hexadecimal character for equipment code. Respecify the equipment codes.	OV	Overflow of volatile storage; appears on the output comment device – no recovery is possible.

TABLE K-4. GENERAL SYSTEM ERROR MESSAGES

Message	Significance	Message	Significance
PARITY, hhhh	Memory parity error at the specified hexadecimal location; appears on the output comment device – no standard recovery is provided. If hhhh = DSA? no parity error was encountered on the core scan. The parity fault was most likely caused by a DSA parity error.	TIMER REJECT	Timer start-up rejected (SPACE or MIPRO)
		STALL REJECT	Stall alarm disable reject (SPACE)
		DISK ERROR (ssss)	Restart the disk to tape program; (ssss) = status
		TAPE ERROR (ssss)	Restart the disk to tape program; (ssss) = status
SET PROGRAM PROTECT	System waiting for program protect switch to be set	GIXX	Ghost interrupt on interrupt line XX reported by LIN1V4.

TABLE K-5. JOB PROCESSOR ERROR CODES

Message	Significance	Message	Significance
JOB ABORTED	The current batch job has abnormally terminated. If the job card included a job name, that name replaces JOB.	JP08 (contd)	to access an unprotected request on a protected unit, or an attempt to select a mass storage device as the standard print unit
JP,yyyyyy	yyyyyy is the last program the library program executed before the job terminated.	JP09	I/O error while accessing the job processor file directory table
JP01,hhhh	Program protect violation occurred at address hhhh.	JP10	Operation attempted on file that is not in the file table; define the file.
JP02,hhhh	Illegal request or parameters at the specified hexadecimal address hhhh.	JP11	File name being defined already exists for another file. Dump the file table to select a name not used previously or attempt a new definition with another name.
JP03, statement	Unintelligible control statement is output with the diagnostic.		
JP04, statement	Illegal or unintelligible parameters in control statement.	JP12	Attempt to access a file that has not been opened
JP05,	Statement entered after manual interrupt is illegal.	KP13	No files are available for definition. Purge the file table to make any expired files available.
JP06	A threadable request was made at level 1 when no protect processor stackspace was available or an unprotected threaded request was made at level 1.	JP14	Attempt to open a previously opened file or attempt to open more than one file on the same unit at the same time.
JP07	Unprotected program tied to access protected device	JP15,xxx	JOB card is not the first control statement in the job or more than one job card is detected within a job. xxx is the control statement in error.
JP08	Attempt to access read-only unit for write, or write-only unit for read, or an attempt		

TABLE K-6. LOADER ERROR CODES

Message	Significance	Message	Significance
E1	Irrecoverable input/output error; terminates load	E10 (contd)	in an * (cr). Core resident entry point tables may also be linked by typing in an *E.
E2	Overflow of entry/external table reservation on mass storage; terminates load	E11	The minimum amount of core is not available for load. At least 195 words plus the length of the loader must be available; terminates load.
E3	Illegal or out-of-order input block; terminates load	E12	Overflow of command sequence storage reservation on mass storage; terminates load
E4	Incorrect common or data block storage reservation. Occurs if the largest common storage declaration is not on the first NAM block to declare common or data storage or, if protected common or data was being used, the NAM block declared a reservation longer than protected common or data; terminates load.	E13	Undefined or missing transfer address; this code is not given if the loading operation is part of system initialization. It occurs when the loader does not encounter a name for the transfer address or the name encountered is not defined in the loader's table as an entry point name; loading is terminated.
E5	Program is longer than area or partitions allotted to it; terminates load.	E14	The loader request operation code word is illegal; terminates load.
E6	Attempt to load information in protected core; terminates load	E15	Overflow of loader table used to store relocatable addresses that have been absolutized to $7FFF_{16}$; terminates load.
E7	Attempt to begin data storage beyond the assigned block; terminates load	E16	Entry point name is not in the loader table; operator must type in the correct entry point name.
E8	Duplicate entry point	E17	Informative diagnostic. Relocatable entry point has been absolutized to location $7FFF_{16}$. If any program in the system is testing for an entry point value of $7FFF_{16}$ to indicate that this entry point is not present, the test is not valid.
E9	High order bit of a relocatable address is set or negative relocation has been encountered during a Part 1 load; terminates load.		
E10	Unpatched externals; external name is printed following the diagnostic. When all unpatched externals have been printed, the operator may terminate the job by typing in an *T (cr) or continue execution by typing		

TABLE K-7. LIBEDT ERROR CODES

Message	Significance	Message	Significance
L01	More than six characters in a parameter name are presented to the library editing program.	L04	Invalid control statement was presented to the library editing program.
L02	More than six digits in a number are presented to the library editing program.	L05	Illegal field delimiter in a control statement was presented to the library editing program.
L03	Improper system directory ordinal was presented to the library editing program.	L06	Illegal field in the control statement was presented to the libraty editing program or I/O was attempted on a protected device.

TABLE K-7. LOADER ERROR CODES (Contd)

Message	Significance	Message	Significance
L07	Errors in loading as a result of a library editing program control statement	L14 (contd)	not longer than the distance from the start of unprotected core to the top of core.
L08	A program to be added to the program library has an entry point duplicating one already in the directory.	L15	Illegal input block encountered; last program stored in library is not complete.
L09	Standard input failed on the first input record following an *N request	L16	I/O input error occurred; last program stored is not complete.
L10	The operator is deleting a program that is not in the library.	L17	An *L program being installed exceeds the capacity of LIBEDT to input from mass storage.
L11	No header record on file input from mass storage	L18	Attempt to load a zero-length program during an *M request or an *N request
L12	On an *L entry statement either there was an input error or the first record was not a NAM block.	L19	No data base entry point specified in the system for use by an *A statement and parameters
L13	Common declared by the program being loaded exceeds available common or system common not specified in the system when requested.	L20	Irrecoverable error occurred during loading
L14	Program being loaded is longer than the size of unprotected core, but	L21	Attempt to write beyond the maximum sector number specified for MAXSEC at initialization

TABLE K-8. COSY ERRORS

Message	Significance	COSY Action
nn ERRORS	This message appears at the end of a COSY job if errors exist. The number specified is the decimal count of errors in the COSY job.	
****COSY Cnn****		
01	First card of revisions deck is not a DCK/, MRG/, CPY/, or END/control card.	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/. MRG/, CPY/, or END/card
02	Illegal parameters on MRG/control card	COSY aborts
03	First card from merge input is not a DCK/ control card.	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/ or END/ card
04	MRG/ control card within revisions decks	COSY aborts
05	Illegal parameters on DEL/, INS/, or REM/ control card	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads the next control card

TABLE K-8. COSY ERRORS (Contd)

Message	Significance	COSY Action
06	Sequence numbers out of order within the revisions set	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads the next control card
07	Two sequence numbers on INS/ control card	Reads revisions and lists them with asterisks in column 1 through 4 until it reads the next control card
08	Control card does not follow DCK/ card when merging revisions.	Reads revisions and lists them with asterisks in column 1 through 4 until it reads next control card
09	First card of source deck not CSY/ or HOL/ control card	COSY aborts.
10	Requested deck not on input library	Read revisions and lists them with asterisks in column 1 through 4 until it reads a DCK/, MRG/, or END/ card
11	Decknames on DCK/ and HOL/ cards do not agree when adding new deck to COSY library.	COSY aborts.
12	Revision card following DCK/ card is not a control card.	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads a control card
13	DEL/ or INS/ card contains sequence number beyond the end of the input deck	Reads revisions and lists them with asterisks in column 1 through 4 until it reads a DCK/, MRG/, or END/ card
14	Illegal parameter on DCK/ card	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, or END/ card
15	Parameter on DCK/ card twice	Uses second parameter
16	DCK/ card requests both H and C or H and L on the same unit.	C or L parameter is ignored; processing continues.
17	DCK/ card requests input from logical unit previously used for output.	Reads revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, or END/ card
18	COSY output requested on unit previously used for Hollerith output or Hollerith output requested on unit previously used for COSY	Illegal output request is cleared; processing continues.
19	Maximum number of output units is exceeded.	Output is cleared; processing continues.
20	The DCK/ card requests output on a logical unit previously used as input.	The output is removed; processing continues.
21	The DCK/ card requests C and L output on the same unit.	The L parameter is ignored; processing continues.
22	The CPY/ control card is not the first card of the revisions deck.	The CPY/ control card is listed with asterisks in the first four columns and the next control card is read.
23	The CPY/ card was not followed by a CPY/ or END/ card.	COSY aborts.

TABLE K-8. COSY ERRORS (Contd)

Message	Significance	COSY Action
L,lu FAILED ec	COSY driver errors are output by the alternate device handler – All errors are catastrophic.	For protected requests, type CU For unprotected requests, type DU
	1 Not assigned	
	2 First record read was not a CSY/ record.	
	3 END/ card was not the last card on COSY input.	
	4 No end-of-file on COSY input	
	5 A read request was made to a logical unit assigned to output or a write request was made to a logical unit assigned to input.	
	6 MOTION request was madeto a logical unit assigned to input/output and no end-of-deck marker was encountered.	
REWIND LUnn	This message may appear at various times during a COSY job. The specified number is the decimal logical unit to be rewound.	Operator must enter any value through the standard input comment device after rewinding the unit.

TABLE K-9. UTILITY PROGRAMS

Message	Significance
IOUP Program	
END OF TAPE LU nnnn ACTION?	An end-of-tape mark is sensed while writing data on magnetic tape. The operator must respond with either $RES, to resume action from the point of last interruption, or $END, to terminate the request.
FILE BACKD FILE nnnn FILE BACKD RECS	The specified unit has been backspaced by nnnn files or records.
FILE SKIPD FILE nnnn FILE SKIPD RECS nnnn	The specified unit has been advanced by nnnn files or records.
FORMAT ERROR	Invalid control statement; re-enter statement.
IN/OUT ERROR LU nnnn	An error occurred in an input/output operation on logical unit nnnn.
MISMATCH REC nn*32768+nnnnn	The indicated record is not the same on both data being verified. Where: nn is 00 through 03; the quotient obtained by dividing the total number of records by 32,768. If nn is 0, only nnnnn is typed out. nnnnn is 0 through 32,767; the remainder obtained by dividing the total number of records by 32,768.

TABLE K-9. UTILITY PROGRAMS (Contd)

Message	Significance
MODE DIFFERENT ON MAG TAPE	One or more records on magnetic tape contain the same information as the record being verified against, but of different mode.
UT FORMAT INCORRECT	The request is not correctly formatted; parameters and/or delimiters are incorrect.
UT INVALID REQUEST	The mnemonic request code is illegal.
SETPV4 Program	
*****ERROR code	SETPV4 error messages are output on the standard list device. Errors occur in two phases: statement reading and statement execution. All errors are fatal, however, some errors may be delayed fatal (DF) allowing all statements to be read and diagnosed. All errors occurring in the statement execution phase are immediately fatal (IF) and cause an exit to the job processor. A flag is set and checked on entry to phase 2 (execution) and, if set, the execution is not initiated.
	Type Error
1	IF An *L control statement must be the first statement.
2	DF Illegal or wrong format for control statement
3	IF An *E must be the last control statement.
4	IF Output is attempted with parameters less than the current position.
5	DF Control statements are out of order (issued after an attempted sort).
6	IF The maximum number of control statements is exceeded (1200 maximum).
7	DF The first statement is an *I or an *R statement and and can not have an asterisk (*) indicating use of the previous binary.
8	IF An attempt is made to access a unit after a file mark has been encountered.
9	IF An *E statement is encountered before an *O statement. Outputting must take place if there are any *R, *I, *D, or *S statements in the set.
10	IF Mass storage overflow
DTLP Program	
ILLEGAL PARAMETERS SPECIFIED	Equipment code specified for disk-to-tape contains a non-hexadecimal character.
LIBILD Program	
INVALID LU	Logical unit illegal
INVALID CLASS CODE	Device is incompatible with the function to be performed.

TABLE K-9. UTILITY PROGRAMS (Contd)

Message	Significance
LAST DECK REJECTED – NOT UNIQUE	There are duplicate copies of the program; program identification must be unique.
LAST DECK REJECTED – NO XFER RECORD	The binary program does not have a transfer record. Type: 1 = Terminate 2 = Proceed to subsequent library 3 = Continue with current library
NAME RECORD NOT 1ST RECORD OF DECK	Type: 1 = Terminate 2 = Proceed to subsequent library 3 = Continue with current library
XFR RECORD MISSING FOR LAST PGM LISTED. PGM DELETED.	Type: 1 = Terminate 2 = Proceed to subsequent library 3 = Continue with current library
TOO MANY BINARY DECKS LOADED. CHANGE LIMIT AND RECOMPILE.	This library has more programs than LIBILD can process.
CHECKSUM ERROR NOTED IN LAST PROGRAM.	Previously generated checksum does compare with the current checksum whrn the program is read from mass memory.
BAD *DEF RECORD. NO IDENT CHARACTER BAD *DEF RECORD. IDENT CHAR ALREADY USED. IGNORED. INVALID DEFINITION RECORD. IGNORED.	*DEF is not the first record of a definition group.
NO DEFINITIONS WERE SUCCESSFULLY LOADED. TOO MANY DEFINITION SETS. IGNORED.	
PROGRAM SPECIFIED BY THIS RECORD NOT FOUND. PROGRAM HAVING THIS ID INFO NOT FOUND. MORE THAN ONE PROGRAM HAS THIS NAME.	The first program on the library with this name will be written to installation file.
ILLEGAL CHARACTER STARTS IDENT FIELD.	Ident field must start with a single quote.
ILLEGAL IDENT FIELD. RECORD IGNORED.	*B record was not terminated by a single quote prior to column 73.
ILLEGAL *B RECORD. RECORD IGNORED.	The name field of *B must be enclosed by single quotes.
NULL PROGRAM NAME. RECORD IGNORED.	The name field consists of two single quotes.
PROGRAM NAME TOO LONG. RECORD IGNORED.	The name on *B contains more than six nonblank characters.

TABLE K-9. UTILITY PROGRAMS (Contd)

Message	Significance
NO DEFINITIONS ARE STORED. RECORD IGNORED.	*USE is encountered, but no definitions are made.
INVALID *USE RECORD. IDENT FIELD. RECORD IGNORED.	No nonblank character was detected prior to column 73.
INVALID *USE RECORD. MAX IMBEDDED LEVEL IS 6. RECORD IGNORED.	This *USE is infinitely recursive.
SKED Program	
INVALID COMMAND	Command is not legal for SKED.
ERROR IN COMMAND FORMAT	A comma, argument, etc. was omitted.
COMMAND NAME NOT UNIQUE	Not enough letters are included to uniquely define the command.
LU NOT LEGAL FOR COMMANDS	LU type is not valid for the command requiring the LU.
SKELETON NOT LOADED	SKELETON was not previously loaded, prior to operation upon it.
RECORD NUMBER IS ZERO	The record number of zero is illegal.
INVALID CHARACTER IN NUMBER	Nondecimal character is specified in number argument.
INVALID RECORD NUMBER	Record number is out of range or the second argument is less than the first argument.
RANGE CONTAINS NUMBER ALREADY DELETED	The record that is referenced has been deleted.
RECORDS HAVE BEEN PREVIOUSLY DELETED	The range of record numbers of the CATLOG command includes numbers that have been deleted.
NO INSERTION RECORD AT SPECIFIED LU	The device defined for insertion records does not contain any records.
RECORDS NOT DELETED PLEASE RESEQUENCE SKELETON	An attempt was made to delete more than 500 records since the file was last resequenced.
RESPONSE MUST BE LU (CR) OR (CR)	An invalid response to the message: ANY MORE INPUT. ENTER LU
EDITOR Program	
FILE NOT DEFINED	File not in job file directory
LINE NUM OVERFLOW	Line number larger than 9999
FILE SPACE FULL	Editor's or user's file full
INVALID COMMAND	Invalid command file full
NAME NOT UNIQUE	Name not unique
INVALID LINE NUMBER	Invalid line number
DIRECTORY READ ERROR	Error while readying the file directory
DISK READ ERROR	Disk error, see device error message

TABLE K-10. FILE MANAGER CODES

Error	Significance	Error	Significance
F.M. ERROR 1	An irrecoverable mass memory error occurred while space was being returned to the space pool. This error may result in	F.M. ERROR 1 (contd)	invalid space pool threads and/or file space being lost to the file manager.

TABLE K-11. FILE MANAGER REQUEST ERRORS†

Error†	Significance	Error†	Significance
0	File defined/not defined	12	Key length not one for indexed-ordered file
1	File locked/not locked	13	Unprotected file request attempt to change a protected file
2	long store or short read	14	File request illegal
3	End-of-file encountered	15	File request rejected; this bit is set whenever: • Bits 14, 13, 12, 11, 10, 8, 7, or 0 are set. • Bit 5 is set for RTVIDX if record does not exist or request is repeated after end-of-link reached. • Bit 4 is set for STOSEQ/STOIDX. • Bit 1 is set for RELFIL, UNLFIL, STODIR, LOKFIL (already locked) RTVSEQ, RTVIDX, RTVIDO, and RTVDIR (attempt to remove from locked file without the combination).
4	At least one more record exists with the same key value		
5	Record does not exist or has been removed		
6	Unused		
7	Mass storage error		
8	No file space left		
9	Attempt to store direct outside File Manager's disk space		
10	File combination incorrect		
11	File already defined/not defined as indexed		

†This is a list of the file request indicator bits. This list is referenced for the reqind parameter in each of the request calls.

On the CYBER 18, the standard system comment device (1811-1 Console Display) can function in a second mode as a computer panel interface. Used in this mode it can be utilized to select processor functions, examine or set memory locations, and examine or set registers. This is accomplished with a processor function control register (FCR) and a set of control function characters. See tables D-1 and D-2.

Functioning as the system comment device, the console display is in console mode. Functioning as the computer interface panel, the console display is in panel mode.

A master clear of the machine places the console display/interface in console mode.

To place the console display/interface in panel mode, press the escape key (ESC) on the console display. The console display accepts a special set of characters in panel mode to perform defined functions.

Some panel functions useful to the MSOS user are described below. For a more detailed description of the panel interface functions, refer to the CYBER 18 Processor with Core Memory Hardware Reference Manual.

When in panel mode, the following operations select processor functions. All numeric values are hexadecimal. After most operations, the FCR register (eight hexadecimal characters) are displayed.

Operation	Function
@	Return to console mode
?	Master clear computer and return to console mode
HG	Halt
IG	Run
I@	Run and return to colsole mode
J28G	Set protect switch on
J28a (@)	Set protect switch on and return to console mode

When in panel mode, the following operations select registers and allow display or setting of the registers.

Operation	Function
J11G	Select P register to K function
KG	Display current value of P
KhhhhG	Set P to $hhhh_{16}$
J14G	Select A register to K function
KG	Display current value of A
KhhhhG	Set A to $hhhh_{16}$
J1BG	Select M (mask) register to K function
KG	Display current value of M
KhhhhG	Set M to $hhhh_{16}$
J03G	Select X register to L function
LG	Display current value of X
LhhhhG	Set X to $hhhh_{16}$
J04G	Select Q register to L function
LG	Display current value of Q
LhhhhG	Set Q to $hhhh_{16}$
J07G	Set L function to current memory location defined by P register
LG	Display current memory location defined by P. P is incremented by 1 after each display
LhhhhG	Set memory location defined by P to the value $hhhh_{16}$. P is incremented by 1 after value is entered

NOTE

When using the 1811-1 console display and the ALERT light comes on, the BREAK key is used to clear the ALERT condition.

TABLE L-1. FUNCTION CONTROL REGISTER (FCR)

Bit	Digit	Bit Definition		Bit	Digit	Bit Definition
31	(LSB)	Overflow	Status Only	15		00 Breakpoint off
30		Protected instruction		14		01 Instruction reference breakpoint
29	7	Protect fault				10 Storage operand breakpoint
28		Parity error			3	11 All references breakpoint
27		Interrupt system active		13		Breakpoint interrupt (breakpoint stop if clear)
26		Auto-restart enabled		12		Micro breakpoint, step, go, stop, (macro breakpoint if clear)
25	6	Micro running				
24		Macro running		11		Step
23				10		Selective stop
22				09	2	Selective skip
21	5	Auto display		08		Protect switch
20		Enable console echo		07		(See table L-2 for parameter selected by the value of this digit)
19		Enable micro memory write		06		
18		Multi-level Indirect Addressing Mode		05	1	Display 1
17	4			04		
16		Suppress console transmit		03		(See table L-2 for parameter selected by the value of this digit)
				02		
				01	0	Display 0
				00	(MSB)	

TABLE L-2. DISPLAY CODE DEFINITIONS

Code	Select Code	Display 1 (K Function)	Select Code	Display 0 (L Function)
0 0 0 0 0	J10:	FCR	J00:	F2 (Addressed by N)
1 0 0 0 1	J11:	P†	J01:	N (MSBs) ††
2 0 0 1 0	J12:	I	J02:	K (LSBs) ††
3 0 0 1 1			J03:	X
4 0 1 0 0	J14:	A (1)	J04:	Q
5 0 1 0 1	J15:	MIR	J05:	F
6 0 1 1 0	J16:	BP /P-MA { Breakpoint Entry	J06:	F1 { Addressed by K Enabled by SM111
7 0 1 1 1	J17:	BP /P-MA { Display only	J07:	MEM

TABLE L-2. DISPLAY CODE DEFINITIONS (Contd)

Code	Select Code	Display 1 (K Function)	Select Code	Display 0 (L Function)
8 1 0 0 0	J18:	SM1		
9 1 0 0 1	J19:	M1	J09:	$\overline{RTJ}$ } Display only
A 1 0 1 0	J1A:	SM2		
B 1 0 1 1	J1B:	M2		
C 1 1 0 0			J0C:	MM
D 1 1 0 1	J1D:	A* } Double Precision Option		
E 1 1 1 0	J1E:	X* } Double Precision Option		
F 1 1 1 1	J1F:	Q* } Double Precision Option		

†Used to address main memory. Automatically incremented after each memory reference.

††Combined contents of these two registers are used to address micro memory. The K register automatically incremented after each memory reference. The N register does not automatically increment.

INDEX

COMMENT SHEET

MANUAL TITLE CDC® Mass Storage Operating System (MSOS) Version 5 Reference Manual

PUBLICATION NO. 96769400 REVISION C

FROM NAME:

BUSINESS
ADDRESS:

COMMENTS: This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number.

CUT ALONG LINE

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 333

LA JOLLA, CA.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
PUBLICATIONS AND GRAPHICS DIVISION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

CUT ALONG LINE

FOLD

STAPLE

STAPLE

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION