96769500

CONTROL DATA
CORPORATION

REAL-TIME OPERATING SYSTEM VERSION 3 REFERENCE MANUAL

CONTROL DATA®

CYBER 18 COMPUTER SYSTEMS

MODELS 10 AND 20

REVISION RECORD

REVISION	DESCRIPTION
A	Manual Released.
(6/76)	
Publication No. 96769500	

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	Revision	SFC[†]
Cover	–	
Title Page	–	
ii	A	
iii	A	
v	A	
vii	A	
viii	A	
1-1	A	
1-2	A	
1-3	A	
2-1	A	
2-2	A	
2-3	A	
2-4	A	
2-5	A	
2-6	A	
2-7	A	
2-8	A	
2-9	A	
2-10	A	
2-11	A	
3-1	A	
3-2	A	
3-3	A	
3-4	A	
3-5	A	
3-6	A	
3-7	A	
3-8	A	
3-9	A	
3-10	A	
4-1	A	
4-2	A	
4-3	A	
4-4	A	
4-5	A	
4-6	A	
4-7	A	
4-8	A	
5-1	A	
5-2	A	
5-3	A	
6-1	A	
6-2	A	
6-3	A	
6-4	A	
6-5	A	
6-6	A	
6-7	A	
6-8	A	
6-9	A	
6-10	A	
6-11	A	
7-1	A	
7-2	A	
8-1	A	
A-1	A	
A-2	A	
A-3	A	
A-4	A	
A-5	A	
B-1	A	
C-1	A	
C-2	A	
C-3	A	
D-1	A	
D-2	A	
E-1	A	
E-2	A	
E-3	A	
E-4	A	
E-5	A	
F-1	A	
F-2	A	
F-3	A	
F-4	A	
F-5	A	
F-6	A	
F-7	A	
F-8	A	
F-9	A	
F-10	A	
F-11	A	
F-12	A	
F-13	A	
F-14	A	
F-15	A	
Index 1	A	
Index 2	A	
Index 3	A	
Comment Sheet	–	
Envelope	–	
Back Cover	–	

[†]Software Feature Change

PREFACE

This manual provides detailed functional descriptions of the external characteristics of the CONTROL DATA® CYBER 18-10 Real-Time Operating System Version 3 (RTOS 3).

It is intended to be used by a programmer-analyst having a basic knowledge of real-time multiprogramming operating systems and familiar with the CYBER 18 Series of computers and assembly language.

The CYBER 18-17 System is equivalent to the old RCOS 2.2 operating system. The Control Data Models 1704, 1714, and 1774 Computer systems are still supported under the CYBER 18-17 System.

Refer to the 1700 MSOS 4 Reference Manual, CDC Publication No. 60361500 for detailed descriptions of drivers, disk addressing, addressing peripheral equipment, magnetic tape recovery, device failures codes, 1536 analog input analysis, 36x communications options, and driver coding structure.

Publication	Publication Number
CYBER 18 Model 17 RTOS 3 Installation Handbook	96769570
CYBER 18 Model 17 RTOS 3 Assembler Reference Manual	96769540
1700 MSOS 4 FORTRAN Version 3A/B Reference Manual	60362000
Magnetic Tape Utility Processor Reference Manual	91637300
Small Computer Maintenance Monitor Reference Manual	39520200
1784 Computer Reference Manual	88830301
CYBER 18 Model 17 RTOS 3 Instant	96769530
CYBER 18 Model 17 RTOS 3 General Information Manual	96769520
1700 MSOS 4 Diagnostic Handbook	60361800
1700 MSOS 4 Macro Assembler Reference Manual	60361900
DRAFT Document Read and Format Translator OCR Software Reference Manual	48143900
CCP Support Software 1 Reference Manual	88988400
RTOS 3 Ordering Bulletin	96769550
CYBER Cross Support System Reference Manual	96836000

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

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

SYSTEM DESCRIPTION 1

The CONTROL DATA® Real-Time Operating System, Version 3 (RTOS 3), is a modular core-resident real-time monitor for the CYBER 18-10 and CYBER 18-20 Computer Systems.

RTOS 3 provides a real-time, multiprogramming environment for user programs in a minimal amount of core. The ability to control execution of user programs, process requests, and handle internal and external interrupts on a priority basis enables RTOS to form the nucleus of a process control, communications, or batch terminal system.

RTOS 3 is compatible with and is based on a subset of the CYBER 18 Mass Storage Operating System (MSOS). Drivers for input/output devices that are provided with MSOS may also be used with RTOS 3. A job processing capability is provided that allows assembly, loading, and execution of nonresident programs. A set of utility programs is also provided.

FEATURES

- The Real-Time Operating System monitor is compatible with the MSOS monitor.
- Any standard MSOS driver may be used with the monitor.
- Interrupt handling is on a priority basis and provides for up to 16 interrupt lines. The status of interrupted programs is automatically saved in the interrupt stack.
- Schedule requests to initiate programs are threaded together and dispatched according to priority. Sixteen program priority levels are provided. There is no limit to the number of programs that may run at the same priority level.
- Input/output requests for a device are threaded by the monitor, according to request priority. The corresponding driver for the device is called to process the request.
- Functional logical unit numbers are assigned to input/output devices. User programs may thus be written independent of the actual input/output device that is used.
- Input/output device errors are recorded in the physical device table for the device. A message is output to the user requesting action. If the error is not corrected, it is reported to the calling program.
- A manual interrupt processor is provided, which interprets and executes operator control statements entered from the comment device. This basic capability allows loading and execution of absolutized programs.
- A system restart routine is provided, which is loaded with the resident monitor, to initialize the scheduler stack, request threads, etc.
- A set of system library programs is provided that can be called by a file name, including the following:

  –Utility assembler (ASSEM)

  –Relocatable binary loader (see section 4)

  –System library editor (LIBEDT) (see section 5)

  –Magnetic tape utility processor (MTUP)

  –Reloadable binary tape editor (SMART)

  –Source tape editor (SETUP)
- Other programs are provided that may be optionally core-resident manual interrupt subprograms providing the following features:

  –Assign logical units to devices.

  –Mark a logical unit up/down.

  –Dump/change core in hexadecimal.

  –Punch absolute record from core.

  –Copy data and/or convert mode.

  –Position magnetic tape.

SYSTEM COMPONENTS

The CYBER 18-10 Real-Time Operating System, Version 3, includes the following modules:

- Monitor (real-time executive)
- Job processor
- Relocatable binary loader
- System initializer
- System maintenance routines

MONITOR

The monitor is the real-time executive of the Real-Time Operating System. It includes the following submodules:

- Internal and external interrupt handler
- Scheduler and dispatcher
- Monitor request entry routine
- Monitor request processors

- Input/output drivers
- Driver find-next-request and complete-request subroutines
- Device error routine
- System data tables

The RTOS monitor is essentially the same as the monitor of the 1700 MSOS, excluding mass-storage features.

JOB PROCESSOR

The loading and execution of nonresident programs are features of the job processor. This is provided by a combination of resident manual interrupt subprograms, which execute operator requests, and system library programs.

RELOCATABLE BINARY LOADER

This program is executed as a job and is used to load, link, and absolutize relocatable binary programs generated by the CYBER 18 utility assembler, 1700 Macro Assembler, or 1700 MS FORTRAN. It is compatible with the MSOS relocatable binary loader.

SYSTEM INITIALIZER

This module is used to install RTOS from the relocatable binary object programs on magnetic tape or binary punched cards. Provision to punch out the absolutized system to magnetic tape is included. The system may also be absolutized in two passes to load the entire available core, if a separate magnetic tape unit is provided for output.

SYSTEM MAINTENANCE ROUTINES

RTOS includes a complete set of programs that are executed as jobs including:

- Utility assembler (ASSEM)
- System library editor (LIBEDT)
- Relocatable binary tape editor (SMART)
- Source tape editor (SETUP)
- Magnetic tape utility processor (MTUP)

MEMORY REQUIREMENTS

RTOS is designed to require the minimal amount of core memory consistent with providing compatibility with MSOS interfaces. Detailed memory requirements for individual programs may be obtained from the CYBER 18-10 and 18-20 RTOS Installation Handbook and/or the program listings.

MONITOR SYSTEM

RTOS for the CYBER 18-10/20 requires variable amounts of core memory for resident programs and data, which include:

- System data (SYSDAT) with expanded monitor stacks and tables
- Monitor, with request processors for Read, Write, STATUS, EXIT, TIMER, SCHDLE, CORE, MOTION, and INDIR
- Load file-by-name routine
- Alternate device handler
- System restart routine
- Calendar and time-of-day routine
- Input/output drivers supporting the conversational display terminal, card reader, line printer, and magnetic tapes

JOB AREA

The job area must have the following minimum size to run the corresponding programs. If a manual interrupt subprogram or other nonresident program is also to be included in the job area at the same time, then core for such programs must be added (approximate):

	Words
Relocatable binary loader, plus four words per entry point, plus the length of program to be loaded	1150
Assembler plus four words per symbol in source (including loader)	4200
System initializer (one-pass mode, see section 7)	3000
System library editor	1300
Relocatable binary tape editor	600
Engineering file list and set	1600

COMPATIBILITY

The RTOS monitor (RTM 3) is essentially a subset of the MSOS monitor, which does not utilize mass storage. The compatibility of RTOS with other CYBER 18 software products is described in the following sections.

MSOS

Software that is core-resident and that runs under MSOS may be run under RTOS, with the following limitations:

- Mass storage features are not provided.
- The system directory structure is not same as MSOS.
- Requests from part 1 (above 32K) are not allowed.
- Preset entry points may not be entered from unprotected memory.
- A core-resident entry point table is not provided (use presets).
- The device error routine does not provide for alternate devices.
- The engineering (diagnostic) file is core-resident and simplified.
- The protect processor (optional) is simplified.
- There is no provision for partitioned core.
- LOADER, GTFILE, SPACE, and RELEAS request processors are not provided.

Any MSOS driver may be added to an RTOS-based system, except those that make implicit use of mass memory. See section 4.

FORTRAN

Programs compiled using CYBER 18 Mass Storage FORTRAN may be loaded by the RTOS relocatable binary loader and executed under RTOS. The core-resident FORTRAN run-time routines may be loaded with the FORTRAN job.

MACRO ASSEMBLER

The utility assembler provided with RTOS accepts the same statements and generates the same code as the 1700 Macro Assembler. However the following capabilities are not provided:

- Macro instructions (MAC, EMC)
- Variable field definitions (VFD)
- Scaled decimal constants (DEC)
- Enable/disable character addressing (ECA, DCA)
- Conditional statements (IFA, IFC, EIF)
- Use of special characters to terminate alpha strings (ALF . , msg.)

CYBER CROSS SYSTEM

Programs assembled and linked using the CYBER Cross System Support Software in a CYBER 70 Computer may be loaded and executed under RTOS.

MONITOR

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, which 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.

INTERRUPT LEVELS AND PRIORITIES

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 at installation time.

Interrupts on any interrupt line are recognized only if (1) the interrupt system is enabled and (2) the bit in the interrupt mask register (M register), which corresponds to the interrupt line, is set to 1. Each interrupt line is assigned a specific priority and an associated interrupt response routine. The priority levels are from 0 (low) to 15 (high). This priority level is an index to a table of interrupt masks (MASKT), which is included in SYSDAT. When an interrupt occurs, the interrupt handler records the current state of the A, Q, I, P, R1, R2, R3, and R4 registers, overflow, and priority level in the interrupt stack. Control is then given to the interrupt response routine at the specific interrupt priority level. After the interrupt has been serviced, the interrupted program can be restored to execution at its original state by restoring the registers from the interrupt stack. When interrupts occur, the hardware recognizes the lowest line number first. Mask table entries for unused interrupt lines are set to 0.

When programs make monitor requests, two priority systems are used: request and completion priority. The request priority determines how a request is to be queued with respect to other requests that are already queued (the higher the request priority, the closer to the front of the queue).

After a request has been completed, the completion priority defines the level at which execution takes place (i.e., after I/O is complete, the completion address is scheduled at the level defined by the completion priority).

Background programs have the lowest priority and are executed at priority levels 0 and 1. Priority declarations from background jobs are ignored. Main processing can be performed at levels 0 and 1, and I/O completion routines are always executed at level 1. All foreground programs should operate at level 3 or above to avoid being affected by the background programs (levels 0 and 1) and the job processor (level 2).

Each driver has a defined operating priority. If a request is made to a driver by a program operating at or above the driver's priority level, the request is not initiated until all previously scheduled or interrupted programs at or above the driver's level have been executed. Conversely, if a request is made to a driver by a program operating below the driver's priority, the requesting program is temporarily suspended and the I/O driver is given control immediately.

MONITOR STRUCTURE

The monitor, in part, is made up of request modules, the common interrupt handler, the dispatcher, and I/O drivers. Figure 2-1 illustrates their functional relationships.

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.

After a request has been queued, control returns to the requesting program if no higher priority program is waiting to run. However, if the program that performs the request has a higher priority, it is executed immediately.

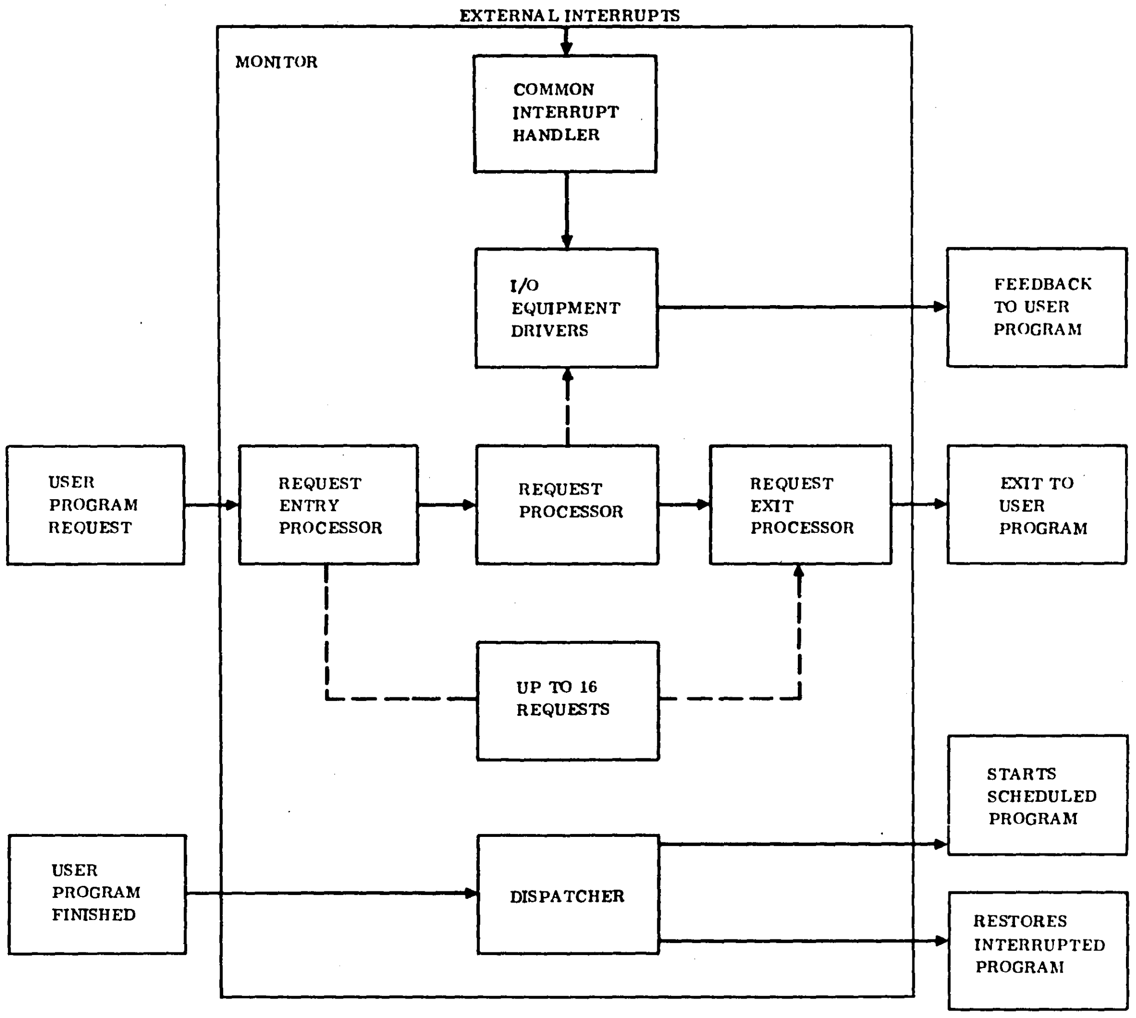


Figure 2-1. Monitor Block Diagram for User Programs

INTERRUPT STACK, EXTENDED INTERRUPT STACK, AND SCHEDULER STACK

The interrupt stack (INTSTK) and extended interrupt stack (EXTSTK) consist 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.

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

	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.

The extended interrupt stack entry contains the following:

	15 … 0
0	REGISTER R1
1	REGISTER R2
2	REGISTER R3
3	REGISTER R4

The scheduler list (SCKSTK) is the waiting list of programs that have been requested by a scheduler request. It is ordered by priority on a first-in, first-out (FIFO) basis within each priority. When a program is taken off the list, 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 list is contained in SYSDAT and is composed of four-word entries. The entry stack contains the following:

	15 … 0
0	SCHEDULER CALL
1	STARTING ADDRESS
2	THREAD TO NEXT CALL
3	Q REGISTER

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 (RCMSCD) will be nonzero. If this overflow occurs some scheduler calls are lost.

The scheduler list is also used for timer requests that are threaded, not with the scheduler calls, but within the counts, tenths, seconds, or minutes threads whose tapes are contained in the timer request processor, TMINT. The entry format differs in word 3, which is used to count down the units of the timer request.

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. Interrupts are only received if the M-register bit, which corresponds to the 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 four-word interrupt trap. The trap region begins at location 100_{16} and ends at location $13F_{16}$.

Each trap entry is of the form:

	15 … 0
0	ENTRY
1	RTJ- (\$FE)
2	p1
3	pp

{RTJ- (\$F8) 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.
P1	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 32K 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 (RCMCIH) 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 information required to save the interrupted program. (See section 2.)

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

The interrupt priority level is established after the required information is saved. Priority 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 (RCMIIP), 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 PROCESSOR

The LIN1V4 interrupt processor (entry point LIN1V4) handles interrupts from the input/output devices on interrupt line 1. When only one device is on line 1, the general interrupt response type is used in lieu of program LIN1V4.

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, LN1TV4, so that devices on line 1 can be identified.

Example:

```
LN1TV4    ADC    P1711
          ADC    P1777R
          NUM    $FFFF          END OF TABLE
```

The interrupt response type described under General Interrupt Processors 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 (RCMIIP) 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. When this fault occurs, a diagnostic message is printed on the comment unit:

```
PW          (power failure)
```

Interrupts are inhibited and the system hangs on a $18FF_{16}$ instruction.

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:

```
PE          (memory parity)
```

Interrupts are inhibited and the system hangs on a $18FF_{16}$ instruction.

Program Protect

The program protect switch must be ON and the optional protect processor (RCMPRO) is required. A program protect interrupt is diagnosed from a test of the program protect skip instruction. When this fault occurs, the protect processor is entered to examine the system conditions and validity of the protect violation. If the violation is legal, an exit is made to the dispatcher to allow execution to continue. Illegal violations cause job termination with a diagnostic message. See section 5.

If a protect fault occurs and the protect processor is not present, a diagnostic message is printed on the teletypewriter:

PF (protect fault)

Interrupts are inhibited and the system hangs on a $18FF_{16}$ instruction.

DISPATCHER

When a program or program element reaches the logical end of its execution, it terminates by jumping to the dispatcher (JMP- ($EA)) or using the EXIT request. The function of the dispatcher is to determine which program is to execute next. There are only two candidates that can be considered: the top entry in the interrupt stack and the top of the scheduler list. The program with the highest priority is placed in execution. If the two programs have equal priorities, the program placed in execution is from the interrupt stack.

When the program to be started is from the interrupt stack, its A, Q, I, R1, R2, R3, and R4 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 program 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 EB_{16} for timer calls) and the mask or M register is set to correspond to the new priority as contained in 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.

RTOS is compatible with the re-entrant FORTRAN scheduler/dispatcher, which may be substituted in nonstandard systems. Refer to the MSOS Reference Manual.

MANUAL INTERRUPT PROCESSOR

The manual interrupt processor (RCMINT) 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.

NOTE

The response to the MI typeout is required to terminate a level 3 loop in MINT. System programs running below 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 associated with manual interrupt job subprograms (see section 5).

Illegal entries cause an ER05 error message.

Entries not preceded by an asterisk cause the scheduling of a named program file from the system library. The Q register contains the location of the input data character string.

Manual interrupts entered while a request is being processed are ignored. However, a manual interrupt request may be processed during execution of a job; i.e., to re-assign logical units (*K) or dump core (*D). After processing the request, J is typed and *C is entered to allow the job to resume execution.

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 D). 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 (lost interrupt), 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 request processor RW determines if the driver is available. If the driver is not busy, its initiator is scheduled, and the request exit processor returns to the caller.

The tape motion requests are handled in a similar way through the T14 motion request processor.

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 is also set in the physical device table and an exit made to the dispatcher.

When the I/O device completes the operation, an interrupt is generated. When the interrupt mask allows the interrupt, the program that is currently executing is stopped, its registers and overflow states are stored in the interrupt stack, and control is given 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 RCMAKQ 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 (RCMFNR)

The find-next-request (RCMFNR) 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

RCMFNR 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, it 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 the highest priority. If no requests are waiting on a device, it exits to the caller at the address of the call plus one.

Device Not Shared

RCMFNR examines the queue to obtain the next request. If none exists, it exits to the caller at the address of the call plus one. If another request is found, it updates the queue, fills the physical device table, and returns to the caller 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 (RCMPQR)

The complete request (RCMPRQ) subroutine is entered by an indirect return jump through $B6_{16}$ 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 RCMPRQ, 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 (RCMAKQ)

The RCMAKQ 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 last valid data into the last word of the caller's buffer in the case of a short data transfer.

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 the number of words is returned to the user's request stack. This allows changes to this parameter by drivers (OCR devices).

INPUT/OUTPUT HANG-UP ERRORS

\n input/output hang-up error occurs when a driver fails to get a completion interrupt on an operation that it initiated.

The diagnostic timer detects hang-ups. The driver establishes a time differential (in increments of seconds) for each input/output operation; upon differential expiration a hang-up is assumed. This differential is entered in the physical device table slot for the device. The diagnostic timer then decrements the time differential each time the module is set into execution. When the differential becomes negative after decrementing, a hang-up is assumed. 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, it transfers control to the alternate device handler. The parameters passed are the logical unit number and error code.

Initially, the diagnostic timer is operated after system start-up. Thereafter, it is periodically reactivated by a TIMER request. The frequency of operation is dependent on a parameter internal to the diagnostic timer program (normally 1 second).

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

DEVICE ERROR ROUTINE

The RTOS device error routine RCMERR is compatible with the MSOS alternate device handler, but does not provide for assignment of alternate logical units and accepts only RP or CU after the error message is typed. The optional RTOS alternate device handler RCADEV may be substituted for RCMERR and is fully compatible with the MSOS alternate device handler.

When a driver detects an irrecoverable failure of an associated driver, 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 set in the Q register.

15 6	5 0
LOGICAL UNIT	ERROR CODE

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

Then the error routine outputs the following message:

```
L,lu FAILED ec ssss
ACTION
```

Where:

lu	is the logical unit number of the failed device.
ec	is the error code.
ssss	is the hardware status (hexadecimal).

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

RP	Directs the request to repeat
CU	Reports the error to the requesting program; the device is allowed to continue processing requests.
CD†	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.
DU†	Activates CU and terminates the current job being processed.††
DD†	Activates CD and terminates the current job being processed.††

Mass storage device drivers do not use the alternate device handler. The comment device must never be marked down, because it is required to bring devices back up once they are operational. Refer to the MSOS Diagnostic Handbook for a complete list of error and status codes. Typical error codes include:

0	Input/output hang-up (diagnostic timer)
1	Lost data
2	Alarm
3	Parity error
4	Checksum error
5	Internal reject
6	External reject

If a downed device is requested by a program, the completion address is always scheduled with an error; i.e., bit 15 is set to 1.

†Not valid for RCMERR; valid for RCADEV

††If the RTOS job cancel routine RCMJCN is resident

This device may be marked up or down using the MARKLU routine (see section 5).

DUMMY DRIVER

The dummy driver is required in RTOS to allow units to be marked down by providing a dummy alternate logical unit.

The dummy driver may be used in systems where it is useful to slew I/O requests by completing the request without error. For example, by assigning the standard print output to the dummy logical unit, physical output is avoided.

MASS-STORAGE RESIDENT DRIVERS

Standard drivers released with MSOS have the capability to operate from core or mass memory residency. RTOS does not allow the drivers to be mass-storage resident.

SYSTEM TIMEKEEPING ROUTINES

The timer package, which is located within the monitor, functions in conjunction with the system hardware timer (for accurate time delays). The system can execute timer requests (RCMT8), compute time of day (RCMTOD), compute date (RCMCAL), and provide auxiliary time/date calendar functions (SETIME).

TIMER REQUEST PROCESSOR

The timer request processor (RCMT8) 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 RCMT8 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.

Example:

1. A program is scheduled for a delay of 10 seconds.
2. The seconds counter within the RCMT8 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 will occur 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 seconds 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 requests are rethreaded to the counts thread for servicing on the next count interrupt.

TIME-OF-DAY PROGRAM

The system time-of-day program (RCMTOD) is a resident program that is initiated during system start-up 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 (SETIME) is used to set and print the system date and time. The program is loaded and executed as a nonresident job and is used to enter a date/time value or to print the current date/time. It may be loaded directly (*L) or by file name SETIME.

The date/time is entered as

```
mmddyyhhmm
```

Entering a carriage return only prints the current date and time without change.

The following is the entry point date information available that is kept in the SYSDAT program:

YERTO	Year (integer)
MONTO	Month (integer)
DAYTO	Day (integer)

AYERTO	Year (ASCII)
AMONTO	Month (ASCII)
ADAYTO	Day (ASCII)

Example:

```
MI
SETIME
ENTER DATE/TIME   MMDDYYHHMM
0401761035
DATE:  1 APR 76    TIME:  1035:00
J
*C
```

If the date/time has not been initially entered, it is printed as:

```
DATE:  ** *** **    TIME:  ****:**
```

The SETIME program requires that entry point AYERTO be a preset entry (see Protected Core-Resident Entry-Point Linkage, section 2) if SETIME is nonresident.

CALENDAR PROGRAM

The system calendar program RCMCAL is a core-resident routine that is called by the time-of-day program at midnight once each day. Its function is to update the date information retained in the SYSDAT data program.

SYSTEM START-UP

The system may be restarted by a master-clear-run operation. This causes the restart routine to be executed. It is assumed that the core-resident system is still intact.

The restart routine is normally core resident at the end of the resident system. The restart routine performs the following tasks:

1. Resets monitor threads, lists, and pointers
2. Sets up the allocatable core area table, located in SYSDAT
3. Protects and unprotects all appropriate core locations and types PP if the proctect processor is included in the system
4. Schedules the diagnostic timer and time-of-day programs
5. Prints a message on the comment device that contains the current system PSR level
6. **Transfers control to the system idle loop**

The following is a typical output on the comment device after system restart:

```
RTOS 3.0   102   010875
```

Where:

RTOS 3.0	is	the system identification (12 characters).
102	is	the system PSR level.
010875	is	the system build date (month, day, year).

To load the absolutized resident system from the system tape (or punched cards), the bootstrap loader must be entered manually into high core and executed. Refer to the CYBER 18-10 and 18-20 RTOS Installation Handbook.

VOLATILE STORAGE

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). 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:

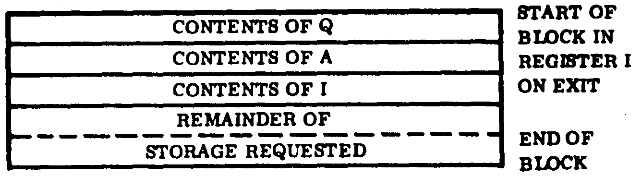
A	Amount of overflow in words
Q	Base address of the interrupt stack

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 a 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.

The standard RTOS size for VOLBLK is 10 levels, with 18 words per level.

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.



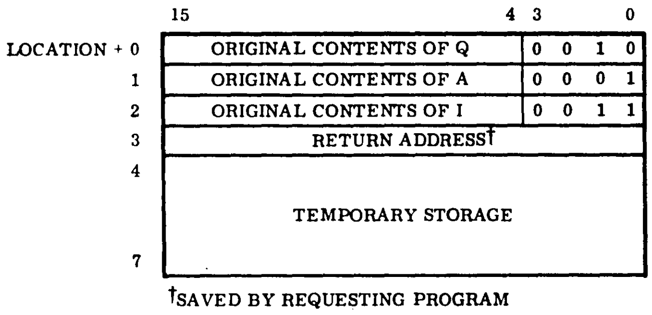
On exit from VOLA, the I register contains the memory location of the contents of the Q register.

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.

```
ENTRY  NUM 0              SUBROUTINE ENTRY
       EQU   VOLA ($BB)
       EQU   VOLR ($BA)
       IIN 0              INHIBIT INTERRUPTS
       RTJ- (VOLA)
       NUM 8
       LDA* ENTRY         GET RETURN
                          ADDRESS
       EIN 0
       STA- 3,I           SAVE IN VOLATILE
         .                PROCESS
         .                CALL
         .
       IIN 0
       LDA- 3,I
       STA* ENTRY         RESTORE RETURN
                          ADDRESS
       RTJ- (VOLR)
       EIN 0
       JMP* (ENTRY)       RETURN, REGISTERS
                          RESTORED
```

On return from VOLA, a block of eight volatile storage locations is assigned.



†SAVED BY REQUESTING PROGRAM

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.

UNPROTECT/PROTECT COMMUNICATION

UNPROTECTED ENTRY POINTS

Programs in protected core may not be entered from unprotected core with the exception of the following:

- Monitor calls – RTJ- ($F4)
- MI subprogram calls – RTJ- ($C0), RTJ- ($EE), RTJ- ($F3)
- Exit to the Dispatcher – JMP- ($EA)

PROTECTED CORE-RESIDENT ENTRY POINT LINKAGE

RTOS provides the MSOS features of allowing a nonresident program to be loaded and linked to core-resident entry points by using the table of presets instead of the MSOS CREP0 and CREP1 tables.

In the system data program, preset entry points are specified by a three-word ASCII name, followed by the address of a protected entry point. Unprotected programs may not call routines that have entry points in the preset table.

This allows nonresident programs to reference the specified resident routines. If jobs are run unprotected, they may only read from the protected resident programs or data tables. Users must specify the preset table by re-assembly of the system data program SYSDAT.

SYSTEM COMMON ORGANIZATION

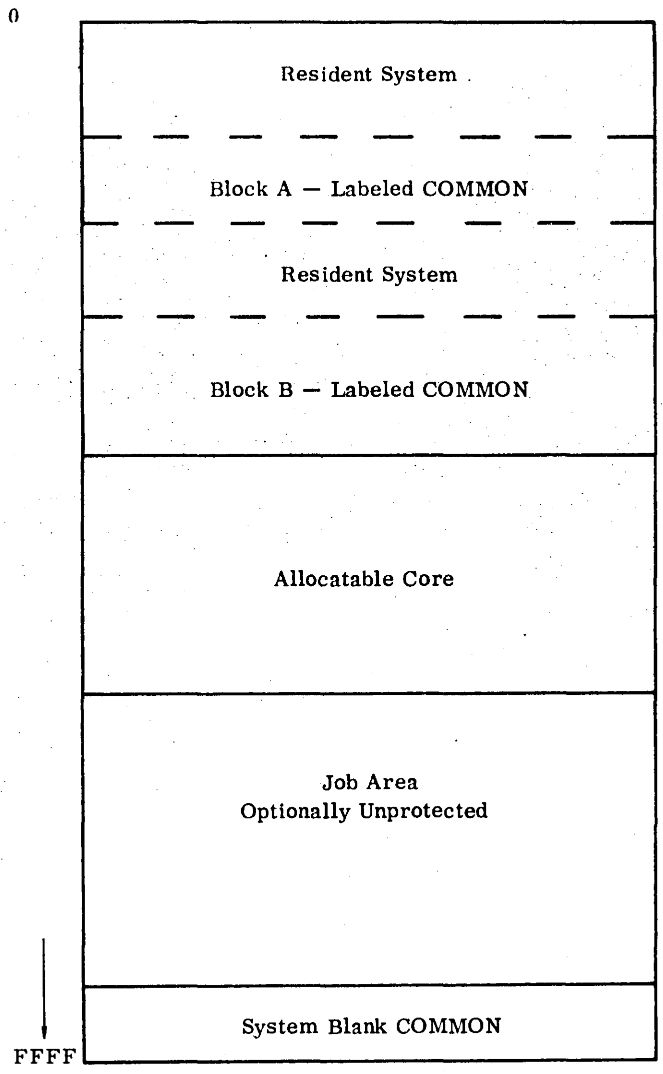
The MSOS provides for use of both blank and labeled common areas. Blank COMMON cannot be preset with data. Labeled COMMON can be preset with a DAT/ORG sequence in assembly language or a block data subprogram in FORTRAN. The system handling of COMMON differs for protected and unprotected programs.

PROTECTED COMMON

For resident 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.

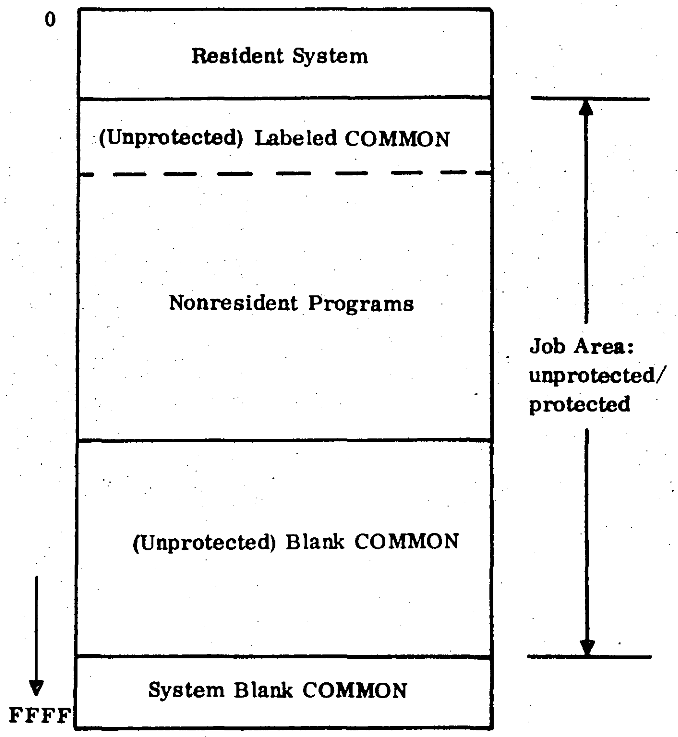
The common block is located in the highest locations of memory.

Labeled COMMON is located in one or more memory blocks and is created during system initialization.



UNPROTECTED COMMON—NONRESIDENT

For nonresident programs, one blank common area and one labeled common area may be specified. These common areas are different areas than those allocated for protected-resident programs. The common areas specified are assigned in the job area with the programs loaded. Blank COMMON is defined at the highest address of the job area. If the protect processor is used, these common areas are unprotected.


0
Resident System
(Unprotected) Labeled COMMON
Nonresident Programs
(Unprotected) Blank COMMON
System Blank COMMON
FFFF
Job Area: unprotected/ protected

MONITOR REQUESTS 3

Requests are used in programs to instruct the monitor to perform operations such as reading, writing, and program scheduling. Each calling sequence contains an indirect return jump to the monitor entry and a list of parameters.

RTOS provides for 16 monitor requests; each has an entry point Txx, where xx is the request code (1 through 16).

The request code is used as an index to a table of request processor entry point addresses. All request processors are entered with the location of the request parameter list in the A register and exit is made by a jump to the request exit entry point. Request codes 0, 7, 13, and 16 through 25 are reserved for MSOS use.

RTOS does not provide the MSOS part 1 (65K) requests. The d bit in the monitor request parameter list is ignored and the Part 1 indirect request is provided so that programs written to run (optionally) in MSOS part 1 also runs under RTOS. RTOS allows absolute, relative, or indirect addresses to be specified.

Table 3-1 lists the requests provided by RTOS that are a subset of those provided by MSOS.

ENTRY FOR REQUESTS

Programs make monitor requests by calling the monitor entry. The calling sequence format is:

```
RTJ-  ($F4)   The fixed communication region loca-
              tion which contains the address of the
              request entry processor.
   .
   .          Parameters
   .
```

THREADING

Requests to the monitor that use input/output drivers or requests to allocate core must wait in a queue for processing. These requests are processed on a priority basis (FIFO within priority).

Example:

The five following requests are initiated by a program.

P,Q,R,S,T	The addresses of the five parameter lists
N	The logical unit specified for each request
E	The physical device table address for the device corresponding to logical unit N (appendix D)
LOG2	The logical unit table (appendix D)

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

In this example, request T has a lower request priority than P, Q, and S. If request T is made before P, Q, and S, it is 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 request 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; that is, it should not loop while waiting for completion by testing the request thread for 0.

NOTE

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.

In figure 3-1, note that the requests are threaded by request priority rather than completion priority. Completion priorities are meaningless if no completion address is specified.

QUEUEING

INPUT/OUTPUT REQUESTS

Input/output requests are queued by logical unit number. Requests for the same logical unit are queued on a thread in the third word of the parameter list. This word contains the first word address of the parameter list of the next request (0 for unqueued requests). The beginning of each queue is identified by an entry in the table of logical units (LOG2), which contains the address of the first word of the parameter list of the first request. The end of the queue is identified by $FFFF_{16}$ in the third word of the parameter list for the last request in the queue.

Unprotected requests may be changed or destroyed between the time parameters are checked and threaded and the time drivers act upon them. To prevent the system from malfunctioning when the parameters are changed, unprotected parameter lists are temporarily protected prior to

TABLE 3-1. MONITOR REQUEST TABLE

Request Code	Request Mnemonic	RTOS/MSOS Request Type	RTOS Usage	Program Handling Request
0		System directory schedule	Not allowed	–
1	READ	Normal read	Allowed	RCMRW
2	WRITE	Normal write	Allowed	RCMRW
3	STATUS	I/O request status	Allowed	RCMT3
4	FREAD	Formatted read	Allowed	RCMRW
5	EXIT	Unprotected exit	Allowed	RCMT5
6	FWRITE	Formatted write	Allowed	RCMRW
7	LOADER	Relocatable loader	Not allowed	–
8	TIMER	Schedule program with delay	Allowed	RCMTMI
9	SCHDLE	Schedule program	Allowed	RCMDSP
10	SPACE	Allocate core	Not allowed	–
11	CORE	Unprotected core bounds	Allowed	RCMT11
12	RELEAS	Release core	Not allowed	–
13	GTFILE	Access files	Not allowed	–
14	MOTION	Tape motion	Allowed	RCMT14
15	TIMPT1	Schedule directory program with delay in part 1.	Not allowed	–
16	INDIR	Part 1 indirect	Allowed	RCMONI
17	PTNCOR	Allocate partitioned core	Not allowed	–
18	SYSCHD	Schedule directory program	Not allowed	–
19	DISCHD/ ENSCHD	Disable/enable directory scheduling	Not allowed	–
20 21 22 23 24 25		Reserved for future MSOS use	Not allowed	–

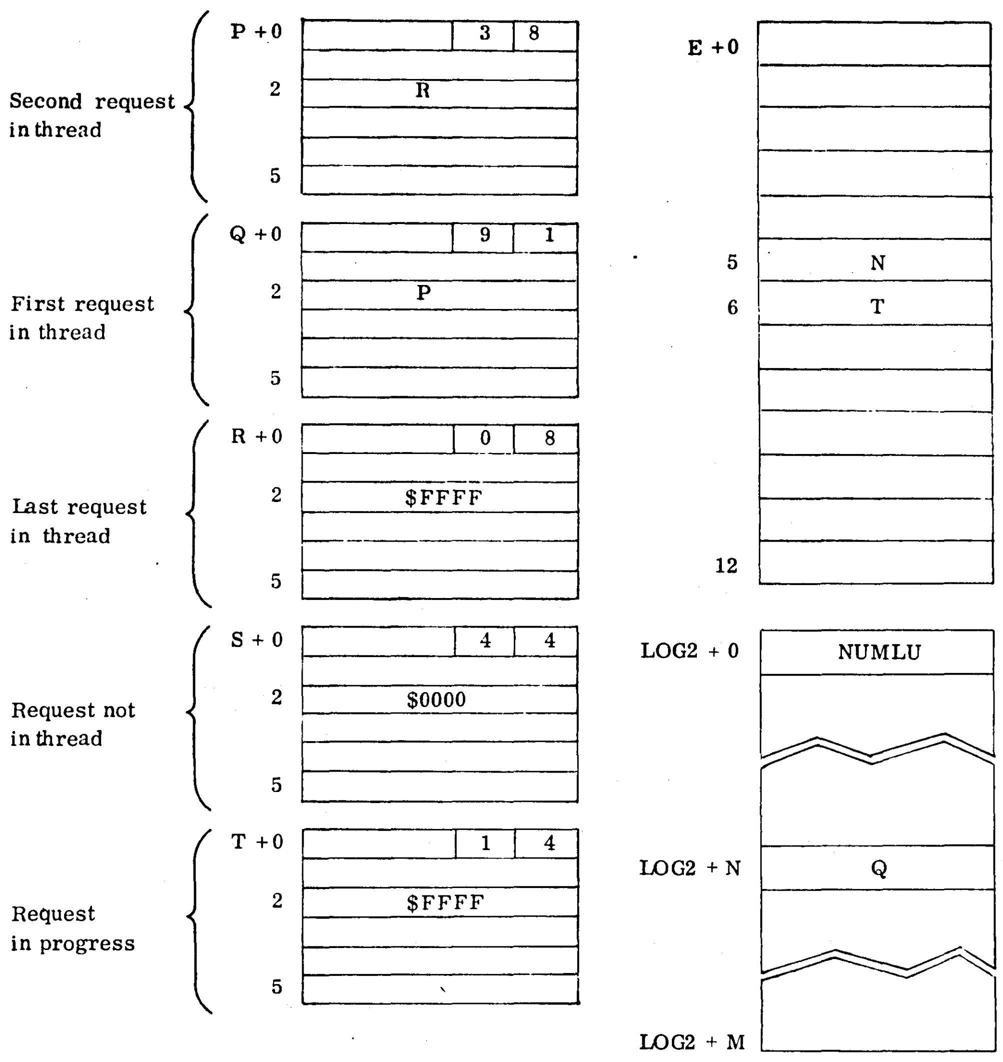


Figure 3-1. Request Threading Layout

parameter checking. The protected parameter list is then threaded normally. On completion (or if the job is terminated), the parameter list is unprotected.

SCHEDULE AND TIMER REQUESTS

Since system programs make many scheduler and timer requests and cannot conveniently wait for a request to be completed before making a new one, parameter lists from these requests are moved to a list and similarly threaded to input/output requests.

After a request has been threaded, control returns to the address following the request. Schedule requests made for a priority that is higher than the current running priority interrupt the requesting program and the scheduled program is executed.

If the list for scheduler and timer requests is filled when a new request is made, the sign bit of Q is set to 1; otherwise it is set to 0.

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

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

REQUEST DESCRIPTIONS

The requests available with RTOS are applicable to both resident and nonresident programs.

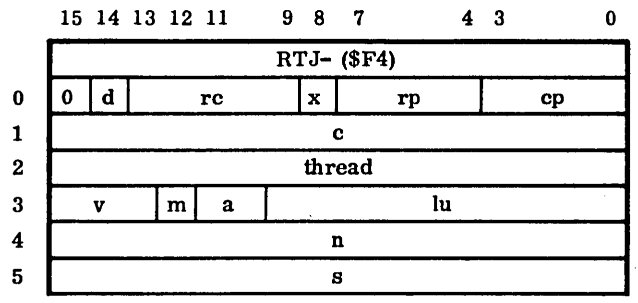
The mnemonic names correspond to the MSOS request macro instructions but RTOS does not support the use of a macro assembler. Otherwise, the request parameters and calling sequences are identical to those for MSOS.

RTOS ignores the d-bit field provided for MSOS part 1 (65K absolute) requests; i.e., RTOS handles requests as if d were set to 0.

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 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 READ/WRITE/FREAD/-FWRITE requests are the same except for parameter n:

rc	The request code
thread	The thread location used to point to the next entry or the threaded list. It must be initially set to 0 and is reset to 0 upon completion.
v	The error code passed to the completion address in bits 15 through 13 of Q and set in the request by the system at completion

Detailed parameter descriptions for the requests are:

lu is the logical unit, an ordinal in the physical device table (appendix D) that may be modified by parameter a.

c is the completion address of the core 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. The notation (c) represents an index to the system library directory, indicating the program to be executed upon completion of the requested I/O operation.

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 the completion status:

15	14	13	Description
0	0	0	No error condition detected by driver; the number of words requested read or written; device not ready
0	0	1	No error
0	1	0	No error; fewer words read than requested
0	1	1	No error
1	0	0	Error condition
1	0	1	Error condition
1	1	0	Error condition
1	1	1	Error condition

When control is returned to the completion address, these bits are set in similar positions in Q and in word 9 of the physical device table (appendix D). If less than n words were transferred on a read, the location following the last word filled is placed in the last word of the user's buffer.

An end-of-file can be verified by checking bit 11 of word 12 in the physical device table via a STATUS request.

NOTE

Use of the v field to designate specific device status (e.g., end-of-tape, successful after recovery, etc.) is dependent on the driver (section 4).

s is the starting address, the address of the first block location to be transferred (see parameter x).

n is the number of words to be transferred.

(n) The 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 0. Some devices signal 0 words as an illegal request.

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

1 Data is converted from ASCII to external form for output, from external form to ASCII for input.

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

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 0 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 1 for unprotected requests. (See Interrupt Levels and Priorities, section 2, for an explanation of priority levels.)

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

0 The first parameter (lu) specifies the logical unit.

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

2 lu is the address of the core location number ($lu \leq 3FF_{16}$).

x is the relative/indirect indicator; this parameter affects parameters c, s, and n as shown here. Because 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 follows the request.
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 follows the request.
≠ 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 device.
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 that is ignored.

Mass Memory Address Format:

15	14 … 0
MSB OF MS ADDRESS	
0	LSB OF MS ADDRESS

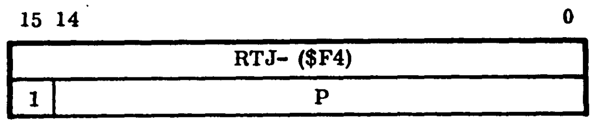
The mass storage address specifies a mass memory word address (READ/WRITE) or a mass memory sector (96-word size) address (FREAD/FWRITE); return is to the location following the mass-storage address.

INDIR REQUEST

The INDIR request allows indirect execution of any other request as determined by the parameter list referenced by p.

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

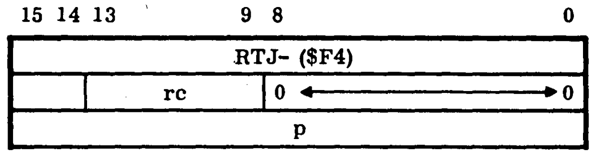
The calling sequence is:



Where:

P is the address of the first word of the parameter list of any other request; bit 15 is set to 1 by enclosing the address in parenthesis.

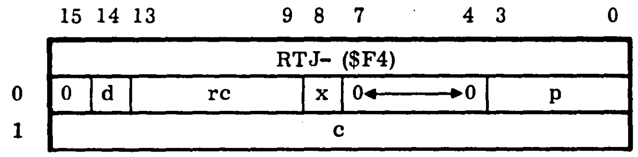
To be compatible with programs that can run in MSOS part 1, an alternate form of the parameter list is provided with the request code 16.



SCHDLE

Programs are queued on a priority basis through the use of the SCHDLE request. 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 or by an EXIT request. The value in the Q register is passed to the requested program on entry.

The SCHDLE request code is nine and the calling sequence is:



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.

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.
	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 that is ignored.

Example:

```
RTJ-  ($F4)
ADC   $1300+LEVEL
ADC   X-*-1
EQU   LEVEL(4)
```

On execution, location X is scheduled at level 4. X is specified relative to the parameter list so that this code may be used in a run-anywhere program.

If a new program is at a priority level higher than the current level, the request is not queued but is immediately executed.

If the program priority level is less than or equal to the current level, the parameter list of the request is moved to a reserved core area for the scheduler list and threaded first by priority and second on a FIFO basis within priority.

The queuing subroutines make entries to the list and the dispatcher removes entries.

When an input/output request is completed, the driver causes the completion routine to be executed by threading the input/output request directly to the scheduler list. This process avoids filling the reserved core area with input/output completion addresses.

The system directory table must be set up by the user if system directory scheduler calls are used. This directory is a simple table of absolute addresses of program starting locations arranged in index order. The entry point name of the first location in the table must be RAT (routine address table), corresponding to system directory index 0. See appendix B.

TIMER

The TIMER request is a delayed SCHDLE request. Through the use of TIMER, a SCHDLE request is made after a specified time delay.

The request code is eight and the calling sequence is:

	15	14	13 9	8	7 4	3 0
	RTJ- ($F4)					
0	0	d	rc	x	u	p
1	c					
2	t					

Where:

c is the completion address to be executed.

p is the priority level of the program.

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.
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 executive 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 that is ignored.

TIMER requests are stacked in the schedule request list but are not threaded with them. Instead, they are threaded together on the basis of time until activation. When the delay for a TIMER request has expired, a SCHDLE request is made with Q to the contents of location E8 (the core clock counter) at the time the SCHDLE request was made. 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.

MOTION

The MOTION control request is used to control motion and end-of-file processing.

The MOTION control request code is 14 and the calling sequence is:

	15	14	13 12 11 10 9	8	7 4	3 0
	RTJ- ($F4)					
0	0	d	rc	x	rp	cp
1	c					
2	thread					
3	v	m	a	lu		
4	r	p1	p2	p3	dy	

Where:

d	is the part 1 request indicator that is ignored.
rc	is the request code.
x	is related only to the completion address.
rp	is the request priority.
cp	is the completion priority.
c	is the completion address.
thread	is the thread location used to point to the next entry of the threaded list. It must be set initially to 0 and is reset to 0 upon completion.
v	is the error code setting.
m	is the mode. A ASCII B Binary
a	is the absolute/indirect indicator for the logical unit.
lu	is the logical unit.
r	is the repeat function indicator (when set to 0).
p_1, p_2,p_3	are the motion control parameters; each of these results in a specific action that 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 0 terminates the request.

TABLE 3-2. RTOS DRIVER ACTION FOR MOTION REQUEST PARAMETERS

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

Key:

MT	Magnetic tape
CR	Card reader
LP	Line printer
CDT	Conversational Display Terminal

dy is the density parameter.

0	No change	
1	800 bpi	External rejects result when an illegal density selection is attempted; e.g., if tape is not at load point.
2	556 bpi	
3	200 bpi	
4	1600 bpi	

One MOTION control request can be repeated for magnetic tape. In this case all of the parameters are the same as in the preceding MOTION request, except for the last word, which is generated as follows:

15	14 – 12	11 – 0
r	p_1	n

Where:

r is the repeat function indicator that is set to 1.

p is the motion code.

n is the number of times to be executed, not to exceed 4,095.

CORE

The CORE request code is 11 and the calling sequence is:

15	14	13 – 9	8 – 0
RTJ- ($F4)			
0	0	rc	0 ←——→ 0

This request is used to set or determine the bounds of the available job area (that portion of core not occupied by a program or data for a job, see section 4). If the A and Q registers are 0 when the request is made, the current upper bound is returned in A and the lower bound 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 the job area, and the upper must be greater than the lower. Illegal values are ignored. Each new request replaces the parameters from the previous request. When an *Z is entered via the job processor, the entire job area is made available again.

STATUS

The STATUS request is used to determine the status of an input/output device by accessing information from the physical device table for the specified logical unit.

The STATUS request code is 3 and the calling sequence is:

	15	14	13 – 11	10	9	8	7 – 0
	RTJ- ($F4)						
0	0	0	rc			0	0 ←——→ 0
1	0 ←——→ 0			a	0	lu	
2	0						

Where:

lu is the logical unit; it is the same as for read/write requests.

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

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

Following execution of the STATUS request, the A register contains the hardware status reply that is word 12 of the physical device table, the Q register contains word 8 of the physical device table (refer to appendix D), and the I register contains the last core address stored on a data transmission (word 11 of the physical device table minus one).

NOTE

Since RTOS 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

The EXIT request is equivalent to a jump to the dispatcher. Control returns to the job processor when the job is completed.

The EXIT request code is five and the calling sequence is:

15	14	13 – 9	8 – 0
RTJ- ($F4)			
0	0	rc	0 ←——→ 0

The EXIT request processor checks for completion of all programs and input/output requests running at levels 0, 1, or 2. If these have all been completed, the processor schedules the job processor, which requests a new control statement and types J, except in batch mode (see section 4). If the EXIT request processor is entered at level 2 or above or if all jobs have not yet completed, it simply exits to the dispatcher.

The protect processor causes unprotected jumps to the dispatcher (i.e., JMP- (EA)) to be handled as EXIT requests. Jobs that are run without using the protect processor must terminate via EXIT requests or control is not returned to the job processor on completion. Jobs that are written according to the rules for manual interrupt subprograms specified in section 4 may exit directly to the job processor on completion.

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 0 or greater than the largest available logical unit number defined in the LOGIA table are illegal.
- Illegal control information – Requests must not contain illegal control information (any information than can cause destruction of part of the system). For example, a READ into protected core is illegal if the calling program is unprotected.
- 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 results in job termination.

SWAPPING CORE

Swapping is a process in which the contents of unprotected core are stored on mass storage to make more unprotected core available for assignment by SPACE requests. This MSOS feature is not provided by RTOS.

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 (not used)	$B3_{16}$	
System library (tape)	$C2_{16}$	

In FORTRAN read/write statements, logical units 1, 2, 3, or 4 indirectly refer to the standard logical units shown above. The operator can change these values via the job processor (refer to section 4). Therefore, all programs can address these particular units indirectly or determine their numbers by interrogating the communications region.

JOB PROCESSOR 4

The Real-Time Operating System (RTOS) provides a simple but effective interface for loading and executing nonresident programs. A job is a nonresident program or a main program and associated subprograms in absolute form that is executed at a low priority level (0 or 1) in the job area of core. This is the core area remaining after all resident programs, allocatable core, and resident common data has been loaded or reserved. The job area corresponds to the MSOS unprotected core area.

PROTECT PROCESSOR

Jobs may be run protected or unprotected. The optional RTOS protect processor, PROTEC, must be resident and the program protect switch must be on to run unprotected jobs. This mode is useful for debugging jobs. Unprotected jobs may only communicate with protected programs via the following:

RTJ-	($C0_{16}$)	Binary to ASCII decimal conversion
RTJ-	($C1_{16}$)	Binary to ASCII hexadecimal conversion
RTJ-	($F3_{16}$)	Manual interrupt processor entry
RTJ-	($F4_{16}$)	Monitor request entry
JMP-	(EA_{16})	Jump to dispatcher

The completion priority in monitor requests originating from unprotected core is forced to level 1 and the request priority to level 0. This provides compatibility with the MSOS protect processor.

The following instructions are treated as do nothing instructions if executed in unprotected jobs:

IIN	Inhibit interrupts
EIN	Enable interrupts
SPB	SET protect bit
CPB	Clear protect bit
EXI	Exit interrupt state

The protect processor is entered, if resident, from the restart program to unprotect the job area, and from the internal interrupt handler if a protect fault interrupt occurs. The condition causing the protect fault is examined. If it is unreasonable, the program that caused the protect fault interrupt is terminated and a diagnostic message is typed:

IRnn, hhhh

Where: nn is the error code (see appendix F).

hhhh is the location of the instruction causing the protect fault.

If the protect switch is off or if the protect processor is not used, jobs are treated in the same way as resident programs by the monitor. No checking is performed. The protect processor is not required to use the RTOS relocatable binary loader, assembler, or other RTOS utility and maintenance routines.

NOTE

The RTOS protect processor is used to execute undebugged user programs. When executing standard RTOS utility routines (library edit, set time/date, etc.), the protect switch must be off so that the routines may store into protected core.

MANUAL INTERRUPT PROCESSOR

Jobs are controlled by statements entered via the manual interrupt processor. When the manual interrupt button is pressed, the processor responds by typing MI and makes a monitor read request to the conversational display terminal. The processor runs at level 3 and hangs in a loop while waiting for input, which suspends job processing.

The operator enters a statement of up to 26 characters, terminated by a carriage return. If this statement begins with an asterisk, then the manual interrupt processor exits to the corresponding manual interrupt subprogram. The address of the manual interrupt input buffer is passed in the Q register to the subprogram.

If the statement entered does not begin with an asterisk, it is assumed to be a file name. The read-file-by-name program RCMLDF is used to find, load, and execute the named program file.

Invalid statements are indicated by the message: ER05. Invalid parameters in statements are indicated by the message: ER04.

The addresses of up to 26 subprogram entry points (MIASUB through MIZSUB) are stored in a vector table corresponding to the initial letter (A through Z) of the manual interrupt entry used to call each subprogram. If the subprogram is not in core, the address is set to 7FFF and the corresponding manual interrupt entries are invalid. Provision is made to load nonresident manual interrupt subprograms and set up the corresponding entry in the manual interrupt vector table.

On completion, the manual interrupt subprogram returns control to the manual interrupt processor, which types J or ERnn if an error was detected and requests a new control statement.

If a job is scheduled into execution, the manual interrupt subprogram exits to the dispatcher. Control returns to the manual interrupt processor when the job is completed.

JOB CONTROL STATEMENTS

All control statements (table 4-1) begin with an * and are followed by a letter that identifies the subprogram entry point.

If the subprogram is not resident, it must be loaded by name or by using the *L command and then executed to place its address in the manual interrupt vector table.

Users may add subprograms for *F, *G, or *H statements. The remaining entries are reserved for expanding RTOS functions. The parameters used in the statements are:

lu	The logical unit
n	The number of files
start	The starting core address
end	The ending core address
base	The base core address (added to the start and end)
hhhh	A hexadecimal value
m	The mode: A ASCII B Binary E EBCDIC
p	The priority level N Not executed
ec	The error code, 0 through 62; 63 or above = all
file	The file number
recs	The number of records

*V directs the job processor to read subsequent control statements from the input device.

*U directs the job processor to read subsequent control statements from the comment device.

*J (or *JOB) enters the job processor at level 0.

MANUAL INTERRUPT SUBPROGRAMS

The manual interrupt processor provides for up to 26 subprograms. These subprograms are called by entering a manual interrupt control statement that begins with an asterisk followed by a letter (A through Z). This letter uniquely identifies the subprogram and is used as an index to a vector table of subprogram entry point addresses.

Manual interrupt (MI) subprograms may be part of the resident system or loaded from the system tape and executed as jobs. If executed as a job and the protect processor is present, the protect switch must be off.

The vector table contains permanent entries for those manual interrupt subprograms that are included in the resident system. Manual interrupt subprograms loaded as nonresident jobs store their entry point address into the manual interrupt vector table. The subprogram can then be called from the manual interrupt processor as if it were resident. When the job is cancelled, the entry point addresses for the nonresident subprograms are deleted. An attempt to execute a subprogram that is not in core causes the manual interrupt processor to type the error message ER05 and exit.

To provide the option of allowing manual interrupt subprograms to be resident or nonresident, the MI subprograms all begin with the following code, which calls the manual interrupt set-up routine at entry point MIENTX.

```
        NAM    progrm
        ENT    progrm, MIxSUB
progrm  ENQ    nn           Manual interrupt
                            vector index
        RTJ-   ($F3)        Call manual inter-
                            rupt setup.
        JMP*   MIEXIT       Exit to the manual
                            interrupt proces-
                            sor, print J.
MIxSUB  NOP    0            Entry from the
         .                  manual interrupt
         .                  processor
         .                  Reserved for *x
                            statement
MIEXIT  ENQ    -1           Print J.
        RTJ-   ($F3)        Exit to the manual
                            interrupt proces-
                            sor.
```

Where:

progrm	is	the manual interrupt subprogram name and file name. It is one to six characters.
x	is	the key character.
nn	is	the index to the manual interrupt vector table. 0 A 1 B . . . . . . 25 Z
MIxSUB	is	the entry point name called from the manual interrupt processor. x is the key character, A through Z.
$F3	is	the location of the address of the manual interrupt processor entry MIENTX, which is used to set up the manual interrupt vector table, then returns control to the subprogram.

TABLE 4-1. JOB CONTROL STATEMENTS

Statement	MI Vector Table Index	Sub-Program	Function
*ADF,lu,n or *ADR,lu,n	0	POSIT	Advance n files/records.
*BSF,lu,n or *BSR,lu,n	1	POSIT	Backspace n files/records.
*C	2	RCMLDC	Continue.
*D,start,end,base	3	DUMP	Dump core.
*EOF,lu	4	POSIT	Write end-of-file.
*F	5	—	User subprogram
*G	6	—	User subprogram
*H	7	—	User subprogram
*I,start,hhhh,hhhh,hhhh,hhhh,hhhh	8	INSERT	Insert 1 to 5 values.
*J or *JOB	9	RCMLDC	Enter job processor.
*K,Ilu,Plu,Llu,Mlu,Clu,Slu	10	ASSIGN	Assign logical unit.
*L,lu,file,hhhh	11	RCMLDC	Load program.
*M,lu	12	MARKLU†	Mark logical unit up.
*N,lu	13	MARKLU†	Mark logical unit down.
*O,lu,ec	14	EFLIST†	List/set code in engineering file.
*P,start,end,base	15	PUNCH†	Punch core.
*Q	16	RSCMEX	Execute SCMM.
*REW,lu or *RWU,lu	17	—	Reserved.
*S,start,p	18	RCMLDC	Schedule
*T,lu,m,lu,m,recs,files	19	COPY†	Copy data.
*U	20	RCMCHG	Read statement from comment device.
*V,lu	21	RCMCHG	Read statement from input device.
*WRON,lu or *WROF,lu	22	MAGSIM	Turn write ring on or off on magnetic tape simulator.
*X	23	RCMLDC	Execute program.
*Y	24	—	Reserved
*Z	25	RCMLDC	Cancel job.

†Subprogram is not resident and must be loaded.

The I register is set to the address of the manual interrupt input buffer upon entry to MIxSUB. The MI subprogram must then decode the ASCII input statement stored in this buffer using the standard conversion routines provided in the manual interrupt subprocessor. The MI subprogram may use the first three words for data storage after initial setup.

Manual interrupt subprograms that require setup of more than one entry are coded as follows:

```
        NAM    MARKLU
        ENT    MARKLU,MIMSUB,MINSUB
MARKLU  ENQ    13        Initial entry
        RTJ-   ($F3)     Set up *M entry.
        JMP*   MARKLX    Go to set up next
                         entry.
MIMSUB  NOP    0         Entry from *M in-
                         put
         .
         .
         .
MARKLX  ENQ    14        Secondary entry
        RTJ-   ($F3)     Set up *N entry.
        JMP*   MIEXIT    Exit to manual in-
                         terrupt processor,
                         print J.
MINSUB  NOP    0         Entry from *N in-
                         put
                .
                .
                .
MIEXIT  ENQ    -1        Print J.
        RTJ-   ($F3)     Exit to manual in-
                         terrupt processor.
        END    MARKLU
```

The jump following the RTJ- ($F3) must be a one-word instruction as MIENTX obtains the address of the manual interrupt entry as the calling address plus 2.

On completion, MI subprograms exit to the manual interrupt processor to get the next control statement. If no error occurred:

```
        ENQ    -1        Print J.
        RTJ-   ($F3)     Exit to the manual
                         interrupt proces-
                         sor.
```

If an error occurred, nn is a two-digit number:

```
        LDQ    =Ann      Print ERnn.
        RTJ-   ($F3)     Exit to the manual
                         interrupt proces-
                         sor.
```

If the subprogram has scheduled a job:

```
        JMP-   ($EA)     Exit to the dis-
                         patcher.
```

More than one subprogram may use the same manual interrupt entry statement provided that they are not required to be in core at the same time. For example, two *F subprograms could be written and stored on the system library with different file names. It is possible to write the subprograms so that the letter(s) following the first letter after the * designates the desired function. For example, entries *BSF and *BSR transfer control to a common routine corresponding to *B. This subprogram checks the fourth character for F (file) or R (record).

NOTE

The RTOS manual interrupt processor executes subprograms at level 3. Jobs initiated via an *X or from the library are executed at level 0. On completion of a job, the optional EXIT request processor RCMT5 returns control at level 0. Thus a request to dump core (*D) is executed at level 3 if it is made immediately following a manual interrupt, or at level 0 if made following completion of a job.

LOAD (*L)

This manual interrupt subprogram is always resident. It is used to load nonresident background and foreground programs. Input must be a single absolutized formatted binary record. The control statement format is:

*L,lu,file,hhhh

Where:

lu	is the input logical unit. If omitted, the standard input device is used.
file	is the number of files to be advanced over (after rewind). If omitted, the input unit is not moved.
hhhh	is the starting address for the load. If omitted, the next available address in the job area is used and incremented by the number of words actually read.

NOTE

Always enter *Z before loading a new program or set of programs to initialize the starting load address to the beginning of the job area. After loading is complete, J is typed. To execute the program and level 0 beginning at the starting address of the load, enter *X. Refer to the Schedule Core Address section below for foreground programs.

ASSIGN LOGICAL UNITS

This manual interrupt subprogram is always resident. The control statement format is:

*K,Ilu,Plu,Llu,Mlu,Clu,Slu

Where:

lu is the logical unit number (decimal).

I is the standard input unit ($F9_{16}$).

P is the standard binary output unit (FA_{16}).

L is the standard list unit (FB_{16}).

M is the output comment unit (FC_{16}).

C is the input comment unit (FD_{16}).

S is the system library unit ($C2_{16}$).

*K is used to assign specific logical units to the low-core cells reserved for the standard system functional units.

The parameters are not ordered but must be separated by commas. Invalid logical unit numbers and attempts to assign read-only units for output or write-only units for input are rejected.

MARK LOGICAL UNITS UP/DOWN (*M, *N)

This manual interrupt subprogram is loaded from the system library by pressing manual interrupt and entering:

MARKLU

The control statement format is:

*M,lu	Mark the logical unit up.
*N,lu	Mark the logical unit down.

where lu is the logical unit number (decimal).

This statement is used to make an input/output logical unit available (up) or unavailable (down) for requests. When a logical unit is marked down, all requests for it are completed immediately with error indication and without attempting to process the request.

MAGNETIC TAPE CONTROL

This manual interrupt subprogram is resident.

The control statement formats are:

*REW,lu	Rewind.
*EOF,lu	Write end-of-file mark.
*ADF,lu,n	Advance n files.
*BSF,lu,n	Backspace n files.
*ADR,lu,n	Advance n records.
*BSR,lu,n	Backspace n records.
*RWU,lu	Rewind and unload.

Where:

lu is the logical unit number (decimal).

n is the number of files or records (decimal).

Requests for invalid logical unit numbers are rejected.

DUMP CORE (*D)

This manual interrupt subprogram is resident.

The control statement format is:

*D,start,end,base

Where:

start	is the starting core address.
end	is the ending core address.
base	is the base core address.

All addresses and values dumped are in hexadecimal.

*D is used to dump the contents of the system output device from the starting to ending address specified. If a base address is specified, it is added to both the starting and ending addresses. If the ending address is not specified, one location is dumped.

The comma preceding the starting address is optional.

INSERT VALUE INTO CORE (*I)

This manual interrupt subprogram is resident.

The control statement format is:

*I,start,hhhh,hhhh,hhhh,hhhh,hhhh

Where:

start	is the starting core address.
hhhh	is the hexadecimal value to be entered.

*I is used to enter one to five hexadecimal values starting at a specified address.

The comma preceding the start address is optional.

PUNCH CORE (*P)

This manual interrupt subprogram is nonresident and may be loaded from the system library by pressing manual interrupt and entering:

PUNCH

The control statement format is:

*P,start,end,base

Where:

start	is the starting core address.
end	is the ending core address.
base	is the base core address.

It is used to output a formatted binary record to the standard punch unit from the buffer specified by the starting and ending addresses. If a base address is specified, it is added to both the starting and ending addresses. If the ending address is not specified, the request is rejected.

The comma preceding the starting address is optional.

COPY AND CONVERT DATA (*T)

This manual interrupt subprogram is nonresident and may be loaded from the system library by pressing manual interrupt and entering:

COPY

The control statement format is:

*T,ilu,imode,olu,omode,nr,nf

Where:

ilu	is the input logical unit number.
imode	is the A, B, E input mode.
olu	is the output logical unit number.
omode	is the A, B, E output mode.
nr	is the number of records (decimal)
nf	is the number of files (decimal).

The mode is specified as:

A	Data is converted from internal ASCII to external BCD or vice versa (seven-track units only).
B	Data is transferred in binary mode.
E	Data is converted from internal ASCII to EBCDIC or vice versa (nine-track units only).

The mode is used to copy data from the standard input logical unit to the standard punch output logical unit until the number of records or the number of files specified is reached or an error occurs. The number of records and files copied is typed on the standard list device.

SCHEDULE CORE ADDRESS (*S)

This manual interrupt subprogram is always resident. The control statement format is:

*S,hhhh,p

Where:

hhhh	is the starting address.
p	is the priority level, 0 to F. If blank, it is level 0.

The specified address is scheduled at the specified level, and control returns to the manual interrupt processor.

NOTE

No reasonability checks are made on this statement.

JOB CANCEL ROUTINE (RCMJCN)

This resident routine is used to terminate jobs running at level 0 or 1. It is called via the manual interrupt *Z statement.

RCMJCN runs at level 3 and executes the following functions:

- Removes any entries in the interrupt stack for level 0, 1, or 2 programs by setting the return address to the address of the dispatcher
- Removes timer and scheduler requests that have completion priorities of 2 or below
- Deletes any pending I/O requests having completion priorities of 0, 1, or 2 and waits for those I/O requests that are in progress to be completed

LOAD NAMED FILE SUBPROGRAM (RCMLDF)

This is a resident subprogram that is called from the job processor (manual interrupt processor). It is used to load a named absolute program file from the system library magnetic tape into core.

The format of the statement is:

pgm,hhhh,p

Where:

pgm	is a four- to eight-character program file name.
hhhh	is the starting address for the load (it may be left blank).
p	is the execution priority.

N	Load but do not execute.
Blank	Use priority level 0.
Xq	Execute and pass q to the program in the Q register.
0-F	Execute at priority level p.

The system library logical unit is rewound and searched for a file header with a name that matches the specified name.

The system library search may be terminated by pressing manual interrupt.

This feature may only be used with a system library tape that was prepared using the RTOS system library editor routine LIBEDT. Refer to section 6.

To load a program file that does not have a file header, use the magnetic tape control subprograms to position the tape; load and execute the program using the *L statement.

Example:

*REW,6	Rewind logical unit 6.
*K,16	Set input unit to logical unit 6.
*L,6,10	Advance 10 files and load from logical unit 6.
*X	Execute

JOB AREA BOUNDS

The job area is defined by the following low-core cells (same as MSOS):

EC	Temporary end of job area plus one
ED	Temporary start of job area minus one
F6	Initialized end of job area plus one
F7	Initialized start of job area minus one

The temporary-start and end-of-job area pointers are used to track the next available core address as successive loads are made into the job area. These pointers are reset to the initialized values by entering an *Z statement.

MANUAL INTERRUPT CONVERSION ROUTINES

The following subroutines are available to manual interrupt subprograms and jobs. Entry is allowed from unprotected or protected programs via an address in low core.

ASCII FIELD DECODE SUBROUTINE (RCMFLD)

This subroutine obtains the next field of an ASCII buffer (hexadecimal or decimal) and produces a binary value. The calling sequence is:

I = Address of ASCII buffer

Q = 1 (decimal) or 0 (hexadecimal)

RTJ- ($EE)

The I register is set to the address that contains the starting field at which processing is to begin. If bit 15 is set to 1, the lower portion of the address is processed; if bit 15 is 0, the upper portion is processed.

The Q register indicates the type of field to be processed (decimal or hexadecimal). If it is set to 1, the field to be processed is decimal; if set to 0, the field to be processed is hexadecimal.

Upon exit from this routine the Q register is set as follows:

Q = -1	Illegal character in the field
+1	A null (FF_{16}) or blank character (20_{16}) was encountered
0	A comma ($2C_{16}$) was encountered

Upon exit from this routine the A register is set as follows:

A = The value in the field converted to binary, or

$FFFF_{16}$ to indicate that there was nothing in the field (, ,), or

The illegal ASCII character that was in the field.

BINARY TO ASCII SUBROUTINE (RCMHXD)

This subroutine converts a binary integer value to a two-word ASCII representation of the decimal value. The calling sequence is:

A = Binary value (0 to 9999)

RTJ- ($C0_{16}$)

On exit, the Q register contains the thousands and hundreds ASCII digits and the A register contains the tens and units ASCII digits.

BINARY TO ASCII HEXADECIMAL SUBROUTINE (RCMHXA)

This subroutine converts a binary value to a two-word ASCII representation of the hexadecimal value. The calling sequence is:

A = Binary value (0 to $FFFF_{16}$)

RTJ- $(C1_{16})$

On exit, the Q register contains the most significant ASCII hexadecimal bytes and the A register contains the least significant ASCII hexadecimal bytes.

EBCDIC CONVERSION SUBROUTINE (CVASEB)

This subroutine is used to convert ASCII data to EBCDIC and vice versa. The calling sequence is:

```
RTJ+    CVASEB
ADC     buffer
NUM     n
```

Where:

buffer	is the address of the data.
n	is the number of words to be converted. To convert from EBCDIC ro ASCII, n must be the complement of the number of words to be converted.

RELOCATABLE BINARY LOADER

The relocatable binary programs produced by the RTOS utility assembler are loaded by the relocatable binary loader. It is loaded as an absolute binary program and relocates itself into the available high locations of the job area. Input is loaded into the available low locations of the job area.

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.

Before input to the loader is read from the standard input device, status is checked on the previous input operation. If error-free, the loader reads the next block; if not, the loader issues a message and may terminate the operation. Loader output consists of memory maps and error messages on the system output comment device.

FEATURES

MEMORY MAP

A memory map, output during the loading process, contains the data and common storage locations (if used) and the name and core location of each program loaded. The map is formatted as follows:

DShhhh	The starting location of data storage (labeled common)
CShhhh	The starting location of common storage (blank common)
xxxxxx hhhh	The name and starting location of the first program
xxxxxx hhhh	The name and starting location of the next program
xxxxxx hhhh	
•	
•	
•	

Where:

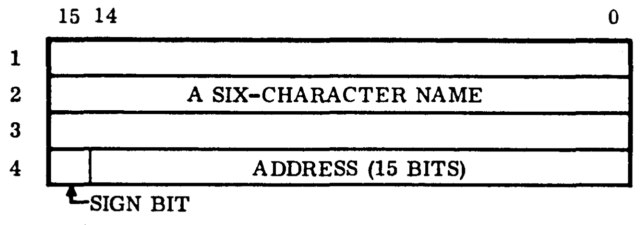
xxxxxx is the program name.

hhhh is the hexadecimal location.

LINKAGE OPERATIONS

The loader carries out two linkage operations: the first when an external name is found to match the entry point name in an ENT input block entry, and the second when an external name is found to match the external name in an EXT input block entry.

During a loading procedure, the loader generates a table of external and entry point names in which each four-word entry is formatted as follows:



The tables for external and entry point names occupy sequential locations toward 0000, immediately in front of the loader. A memory location within the loader contains the length of the table, and the tables are expanded backward in storage. The table length is limited only by the amount of core required for command sequence storage.

Characters form a name from a seven-bit ASCII code. Bits 8 and 15 of each word containing part of a name are not used. If the sign bit of the fourth word is:

0	Entry is an entry point name and address.
1	Entry is an external name and link address.

Link addresses are pointers that string together instructions using the same external name. The 15-bit address of each instruction using an external name points to the 15-bit address of the previous instruction using that name. The address of the first instruction in the program to use this external name is placed in the loader table as that external name's link address.

When the loader matches an external name with a previously loaded entry name, each location containing a link address for the external name is replaced. Within the original source program, there are four ways to reference an external name in the address field of instruction:

Absolute and direct

Absolute and indirect

Relative and direct

Relative and indirect

The type of linkage is determined by the use of the external name in the original source language. If it is used with either type of absolute addressing, bits 14 to 0 of each location containing a link address for this name are set to the value of the entry point address for this name. Bit 15 of this location is not changed by patching in absolute values. If the external name is used with relative direct or indirect addressing, bits 15 to 0 of each location containing a link address for this name are set to the signed value of the increment, which must be added to the program counter to determine the entry point address.

Linkage is the same at every location containing a link address for a given external name. The entry in an EXT block containing this external name determines whether it is for relative or absolute addressing. If the first word of this entry has a sign bit of:

0	Linkage is for absolute addressing.
1	Linkage is for relative addressing.

TRANSFER ADDRESS

After an EOL block is processed, the loader leaves the address of the last transfer name in location E4 of the communications region.

DATA AND COMMON DECLARATIONS

The RTOS relocatable binary loader absolutizes programs beginning at the start of the job area. The labeled common data (DAT), if any, is followed by the command sequence storage. Blank common data (COM) is assigned at the high end of the job area and is overlaid by the loader itself.

RELOCATABLE BINARY INPUT

Blocks of ASCII are identified by an asterisk (00101010) in bits 15 through 8 of the first word. If these bits are anything else, relocatable binary is assumed.

The driver verifies that the block is read correctly for the input device.

The loader recognizes relocatable binary blocks by the type 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

If the loader is unable to recognize the indicator, it does not process the 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.

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 (00101010), 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 BLOCKS

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.

LOADING RELOCATABLE PROGRAMS

First load and execute the relocatable loader using the *L and *X statements or by entering the file name LOADER.

The loader moves itself in the highest available locations of the job area (except for the *XN option), and the relocatable binary programs are absolutized in the lowest available locations of the job area. The loader entry point table is built back from the start of the loader.

When the loader has been read into core, the job processor types J. To set up the relocatable binary programs on the input device, enter one of the following:

*X	Load and execute
*XL	Load, save absolutized program, and execute
*XA	Load and save absolutized program
*XR	Load only (for library edit)
*XN	Load and save absolutized program; loader remains in lower core

When a file mark or EOL block is encountered during loading, an end condition results. When an EOL (*T) block is encountered, loading is complete. When a file mark is sensed, FM is typed by the loader.

The operator may continue loading by putting another program on the system input device and typing a carriage return, or type *T(cr) to inform the loader that loading is complete.

When loading is complete, any unlinked externals are typed. The program is then written on the binary output device (if the *XL, *XN, or *XA option was chosen). The job is executed automatically (if *X or *XL was chosen) or control is returned to the system (*XA or *XN).

LOADER ERROR DIAGNOSTICS

A loader error occurs if the addressing modes for referencing the external name EXTNAM are not the same in both PROGA and PROGB. An error indication is given by the loader, and the loading operation terminates.

Error messages appear on the console immediately following the name and base address of the relocatable binary program loaded. For unrecoverable errors, the loader terminates operation following the error printout. Refer to appendix F.

SYSTEM MAINTENANCE AND UTILITY ROUTINES 6

SYSTEM LIBRARY EDITOR (LIBEDT)

The library editor is a run-anywhere program that is designed to build and maintain absolute library tapes. When used with the job processor and/or relocatable binary loader, the system library editor has the following capabilities:

- Library creation
- Adding to the library
- Library maintenance
- Copying the library or parts of the library
- Listing the library

Programs contained on library tapes may be loaded by using the same file name as specified for the loading of a named absolute program file (see Load Named File Subprogram (RCMLDF) in section 4).

LIBEDT may be used with one magnetic tape drive for creating, adding, or listing the library. Functions such as copying or deleting require two magnetic tape drives.

The library editor executes under the RTOS job processor and is used with the relocatable binary loader. The primary function of the editor is to process and create library tapes containing labeled files that correspond to absolute load modules. The job processor requires the labeled file structure if file names are used to load programs. Labeled files may be intermixed with unlabeled files. Care must be taken to properly position such files before using LIBEDT.

LIBRARY TAPE STRUCTURE

The structure of library files on the library tape is illustrated in the following figures. Figure 6-1 illustrates a system library including the RTOS system loader and an image of the saved resident system (output by the system initializer).

Note that absolute program module 4 is not preceded by a file header label. This is proper since module 4 cannot be loaded with a load-named-file (RCMLDF) statement. *L must be used instead. Also, module 4 cannot be deleted or replaced by LIBEDT.

When the library tape is listed, module 3 is listed as unlabeled.

Figure 6-2 illustrates a user library that does not include the RTOS system.

Program files containing header labels may be loaded by file name or by statement. Program files that do not contain file header labels cannot be loaded by file name. All files are recorded in formatted binary.

FILE HEADER LABEL STRUCTURE

The file header label contains information necessary for program loading and maintenance. The structure of the header label is similar to the ANSI standard file label HDR1.

The fields in the file header label (HDR1) are arranged as shown in table 6-1.

The file header label generated by the system library editor contains appropriate values as defined in all optional fields. Only fields 1 through 8 and 11 are used by the job processor and the system library editor.

OPERATIONAL PROCEDURES

The system library editor creates or updates a library tape. The list of functional procedures included is:

- Add a program to the library or create a new library containing a program.
- Copy the library or portions of the library to a new library.
- Change standard logical units for the system library and punch unit with LIBEDT.
- Get a program from LIBEDT. The specified file is loaded and may be used with an add function.
- Set the values of the first word load address, last word load address, and load transfer address.

The system library editor is generally used with the system loader or relocatable loader when adding programs to the library tape. The program to be added must be loaded by the relocatable loader prior to executing LIBEDT or loaded via the get function by LIBEDT. Since the library editor is run-anywhere, it should be loaded above the program in memory. Normal loading by file name ensures that LIBEDT is loaded at the top of memory.

The system library editor may be loaded from the RTOS 3 system library by name as follows:

Press the manual interrupt button.

The system types:	MI
Enter:	LIBEDT
The system types:	LIBEDT
	IN

The message IN implies that the library editor is ready to accept a function. These functions are described in the following sections.

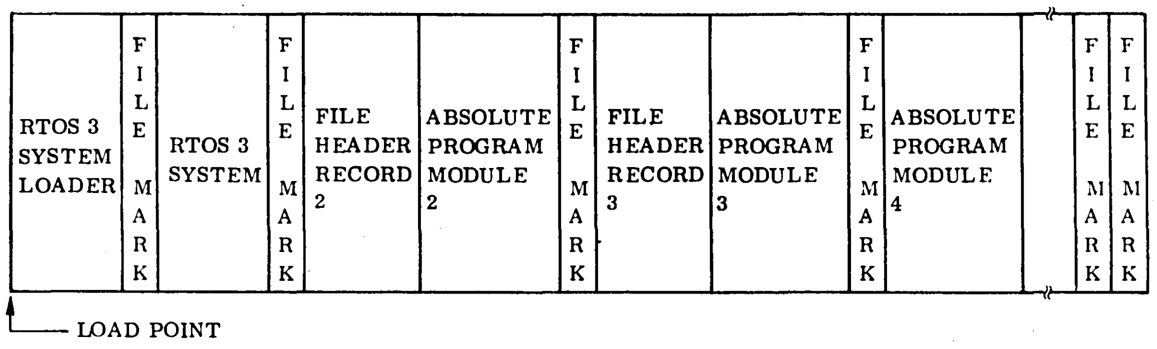


Figure 6-1. Library Tape Structure

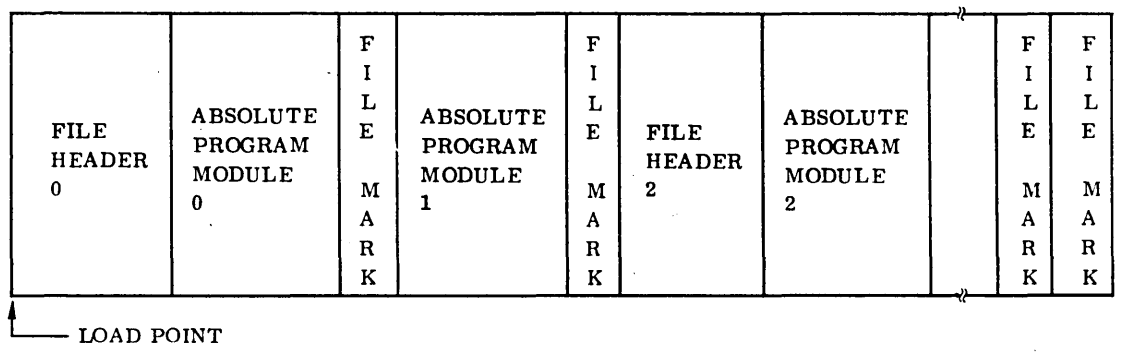

FILE HEADER 0
ABSOLUTE PROGRAM MODULE 0
FILE MARK
ABSOLUTE PROGRAM MODULE 1
FILE MARK
FILE HEADER 2
ABSOLUTE PROGRAM MODULE 2
FILE MARK
FILE MARK
LOAD POINT

Figure 6-2. User Library Structure

NOTE

Leading asterisks (*), if any, are ignored in the control statements.

ADD AND ADDP STATEMENT

The ADD or ADDP statement instructs the system library editor to add the absolute program currently in memory to the library tape or scratch tape. The tape receiving the new program must be mounted on the standard binary output device.

The format of the add statement is

ADD,filename,DATE=yyjjj,Pnn,RA

ADDP,filename,DATE=yyjjj,Pnn,RA

Where:

filename	is a one- to eight-alphanumeric character file identifier, beginning with an alphabetic character from A to Z. Each file name on the library must be unique.
DATE=yyjjj	is the current date in Julian format.
	yy is the current year.
	jjj is a three-digit Julian day.
Pnn	is the running priority for the program.
	nn is a priority specification from 0 to 15.
RA	is a run-anywhere program.

If the ADDP format is specified, LIBEDT positions the library tape past the last file, before writing the absolute program. If the ADD format is used, LIBEDT does not position the library tape before writing the absolute program. The user is cautioned to properly position the library tape if the ADD format is used. Both the ADD and ADDP functions terminate the library by writing two file marks. The tape is positioned before the second file mark, permitting subsequent add or copy functions.

COPY STATEMENTS

The copy statement allows a library tape to be copied to a new tape. It requires two tape units and is designed to

TABLE 6-1. FILE HEADER LABEL FORMAT

Character Position	Field	Name	Length	Description
1 to 3	1	Label identifier	3	Must be HDR
4	2	Label number	1	Must be 1
5 to 12	3	File identifier	8	Any alphanumeric characters; the first character must be alphabetic. This field is used to identify the file.
13, 14	4	Load address	2	Four hexadecimal digits specifying the load address of the first word. If the load address is 0, the program is assumed to be run-anywhere.
15, 16	5	Length	2	Four hexadecimal digits specifying the length of the program module
17, 18	6	Priority	2	Two hexadecimal digits specifying the priority at which the program runs
19, 20	7	Transfer address	2	Four hexadecimal digits specifying the first word address to receive control if different from the load address
21	8	Reserved	1	Blank
22 to 27	9	Set identification	6	Any alphanumeric characters to identify this set of files. This identification must be the same for all files of a multifile set (optional).
28 to 31	10	File section number	4	The file section number of the first header label of each file is 0001. This applies to the first or only file on a volume and to subsequent files on a multifile volume. This field is incremented by one on each subsequent volume of the file (optional).
32 to 35	11	File sequence number	4	Four numeric characters denoting the sequence (i.e., 0001, 0002, etc.) of files within the volume or set of volumes. This field contains the same number (optional) for all labels of a given file.
36 to 39	12	Generation number	4	Four numeric characters denoting the current stage in the succession of one file generation by the next. When a file is first created, its generation number is 0001 (optional).
40, 41	13	Generation version	2	Two numeric characters distinguishing successive iterations of the same generation. The generation version number of the first attempt to produce a file is 00 (optional).
42 to 47	14	Creation date	6	A space followed by two numeric characters for the year, followed by three numeric characters for the day (001 to 366) within the year.

TABLE 6-1. FILE HEADER LABEL FORMAT (Continued)

Character Position	Field	Name	Length	Description
48 to 53	15	Expiration date	6	Same format as field 9. This file expires when today's date is equal to or later than the date given in this field. When this condition is satisfied, the remainder of this volume may be overridden. To be effective on multifile volumes, the expiration date of a file must be less than or equal to the expiration date of all previous files on the volume (optional).
54	16	Accessibility	1	An alphanumeric character that indicates restrictions on who may have access to the information in this file. A space means unlimited access; any other character means special handling, in a manner agreed between the interchange parties (optional).
55 to 60	17	Block count	6	Must be zeros.
61 to 73	18	System code	13	Thirteen alphanumeric characters identifying the operating system that recorded this file (optional).
74 to 80	19	Reserved for future use	7	Must be spaces

facilitate library maintenance and the creation of special forms of the library. There are three formats that perform unique functions.

All variations of the copy function copy files from the system library unit to the standard system output unit. The output tape is terminated by two file marks, and on completion of the function, the tape is positioned between the two terminating file marks. This positions the output tape for subsequent add or copy functions and properly terminates the library.

The format 1 copy function is:

```
    *COPY
or  *COPYB
```

It is designed to copy one file from the system library unit to the standard system output unit. The input tape is copied from the current position to a file mark. The output tape is terminated by a double file mark.

The COPYB alternative causes the output tape to be positioned backwards one file before initiating the copy. When used with the get and add functions, COPYB can be used to create a library file consisting of a file name, the absolutized loader, and the relocatable binaries of a program without intervening file marks.

The format 2 copy function utilizes the file name specified to terminate a copy operation. Its format is:

```
    *COPY,filename
```

Since the copy is controlled by file name, the input tape is rewound prior to initiating the copy. Thus, all files from the beginning of the input tape are copied up to, but not including, the file specified or, if not found, to the terminating double file marks.

The format 2 variation permits selective library copying used in library maintenance. The format 3 copy function is designed to copy selectively from the first file name specified up to, but not including, the second file name specified. Its format is:

```
    *COPY,filename,filename
```

The input tape is rewound prior to copying. The output file is not positioned before copying. To copy a library from a specified file name to the end, a dummy file name should be specified for the last file name. For example:

```
    *COPY,LIBEDT,DUMMY
```

where DUMMY is not a file on the input library.

GET STATEMENT

The get function of the system library editor is designed to load a named file from the system library unit. Once loaded, the file may be added to the output unit via the add function. The format of the GET statement is:

```
    *GET,filename
```

The get function may be used to load a file that is to be added to another library or the same library. If the file is added to the same library, then a different file name must be specified in the add statement. Thus, the get and add functions may be used to change the name of a file. (The original file can be deleted by copying all files except the file to be deleted.)

GET is used to create and maintain the system installation tape. In addition, proper use of the get, add, and copy functions permit the creation of a file that contains the file name, relocatable loader, and relocatables. Although greater in size, a file of this type may be loaded and executed in any area of memory.

The system library editor utilizes the standard system logical units for the system library, system output, and system list devices. The *K statement allows the user to change the standard system library unit and/or the standard system output unit. The *K statement pertains to LIBEDT only and does not alter the system unit assignments. The format of the *K statement is:

```
*K,Snn,Pnn
```

Where:

S is the system library unit.

P is the system output unit.

Either S or P or both may be specified.

LIST STATEMENTS

The LIST statement produces a listing of all the file headers on a library tape. This statement does not require any parameters; its format is:

```
*LIST
```

The library tape must be mounted on the standard system library unit. The list of file names is generated on the system list device. The format of the listing is illustrated below:

```
FILE  NAME    LFWA  LTRA  LLEN  PRT   CDATE
0000  LIBEDT  0000  0000  0499  00    75031
0001  LOADER  0000  0000  04AF  00    75031
0002  ASSEM   0000  0000  04AF  00    75031
0003  SMART   0000  0000  04AF  00    75031
0004  SETUP   0000  0000  04AF  00    75031
0005  POSIT   0000  0000  0072  00    75031
0006  ASSIGN  0000  0000  0031  00    75031
0007  INSERT  0000  0000  0021  00    75031
0008  PUNCH   0000  0000  0045  00    75031
0009  MARKLU  0000  0000  0036  00    75031
0010  DUMP    0000  0000  0117  00    75031
0011  COPY    0000  0000  0199  00    75031
0012  SETIME  0000  0000  04AF  00    75031
0013  EFLIST  0000  0000  04AF  00    75031
```

The column labeled LFWA represents the first word address where the file is to be loaded. If LFWA is 0, then the file is considered run anywhere. The column labeled LTRA represents the execution (i.e., loader transfer) address. That is, after the program is loaded, control is transferred to the loader transfer address. If the LFWA is 0 then the LTRA represents a positive increment that is added to the load address computed by the system.

The column LLEN represents the length of the first record following the header. When the file is run anywhere, the load length is used to compute the load address. Run-anywhere programs are loaded at the top of the system minus the length. Column PRT represents the execution priority and the CDATE represents the Julian creation date.

Because the relocatable binary program file consists of the file header followed by the absolutized relocatable binary loader followed by the actual relocatable binary programs, the length in column LLEN is the same as for the loader.

SET STATEMENT

The set statement allows loader locations to be changed. Locations E4 and ED are set by the loader after loading to correspond with the following (location F7 is not changed by the loader):

2 First word address of load

3 Last word address of load

4 Transfer address

The system library editor can be used to change these values via the set command.

The format of *SET is:

```
*SET,Fxxxx,Lxxxx,Txxxx
```

Where:

xxxx is a one- to four-digit hexadecimal number.

F is the first word address of the load.

L is the last word address of the load.

T is the transfer address.

Any one or all of the above may be specified in a set command.

The set command is intended primarily for debugging. In general, the set function is utilized when erroneous information is to be corrected or special circumstances require it.

LIBEDT ERROR MESSAGES

LIBEDT error messages describe the failure of a LIBEDT function. Appendix F includes a list of the error codes and their descriptions.

RELOCATABLE BINARY TAPE EDITOR (SMART)

SMART (system to maintain a relocatable tape) allows the user to update and maintain tapes containing relocatable binary programs. Control statements are entered via the conversational display terminal.

SMART STATEMENTS

SMART performs the input/output functions listed in table 6-2. An n in the table refers to a decimal digit and x refers to a character in a symbolic name. Each statement must be followed by a carriage return and a space precedes the parameter.

OPERATOR MESSAGES

If an invalid control statement or an inapplicable parameter is used, ? ? is printed and control is returned to the user for correction. Only the EX statement allows control to return to RTOS.

When a file mark is sensed, SMART backspaces over the file mark, halts the tape, prints FILE MARK, and returns control to the user. A file mark does not necessarily mean an error condition. The user may wish to backspace over a file mark to permit the addition of programs within a file.

SOURCE TAPE EDITOR (SETUP)

SETUP (source edit tape update program) provides the capability of creating and updating a magnetic tape file containing one or more assembly language or DRAFT source programs. It is assumed that the reader is familiar with assembler source statements (refer to the RTOS Utility Assembler Reference Manual) and/or with DRAFT source statements (refer to the DRAFT Reference Manual).

SETUP operates under control of RTOS and is loaded as any other binary program. Upon execution, the processor name SETUP is displayed.

DESCRIPTION

Two types of statements are accepted by SETUP: assembly language or DRAFT source statements and SETUP control statements. All source statements are formatted and written on the output master file. All control statements are processed as such and never appear on the output tape.

Source statements conform to the format defined in the RTOS Utility Assembler Reference Manual or in the DRAFT Reference Manual.

The SETUP control statement functions are grouped as follows:

- Those used for assigning input/output devices: master-in, master-out
- Those used for update purposes: delete, insert, and end update
- Those used as file control functions: close, copy, position, list, sequence, no sequence, and exit

SETUP STATEMENTS

Source language statements are accepted by SETUP as free-form statements and are formatted into fixed-form records for output on magnetic tape. During an update, all source language statements input from magnetic tape must be in fixed form.

Free-Form Statements

Free-form statements, input from the conversational display terminal, are formatted into fixed-form statements prior to input. One or more blanks are used to separate the location, operation code, address, and comment fields.

All information preceding the first end-of-field indicator is left-justified into the location field. Data between the first and second end-of-field indicators is left-justified into the opcode field. Data between the second and third end-of-field indicators is left-justified in the address field. All remaining data up to the end-of-statement indicator is left-justified in the comments portion of the record. One or more spaces is required to separate the address field from the comments.

An asterisk appearing as the first character of a statement is considered, along with the remainder of the statement, as a comment. The only formatting performed is left justification into the output record, blank fill, and the addition of identification and sequence numbers.

Input Statements

Input statements have the following characteristics: a blank is the end-of-field indicator for the location, operation code, and address fields; consecutive blanks are treated as a single blank except within the comments field; and a carriage return is the end-of-statement indicator. For example:

```
LABEL OPCODE ADDRESS COMMENT
```

Thus, ORG $20D0 indicates a blank location field while TAG ORG $20D0 illustrates the use of a location symbol.

Fixed-Form Statements

Each free-form statement is output as one fixed-form, 80-character record consisting of the following:

Characters	Description
1 to 6	Location Field
8 to 13	Operation Code

TABLE 6-2. SMART FUNCTIONS

Format	Function	Description
MI∧nn †	Master Input	Assigns the input tape to the logical unit specified by nn. If this statement is not used, input is assigned to the standard input device.
MO∧nn	Master Output	Assigns the output tape to the logical unit specified by nn. If this statement is not used, output is assigned to the standard output device.
PI∧xxxxxx	Position Input	Position input tape forward until the program name xxxxxx is encountered. If a file mark is encountered before the specified program name is found, the file mark message is displayed.
PO∧xxxxxx	Position Output	Position output tape forward until the program name xxxxxx is encountered. Functions are identical to those of position input.
CP∧xxxxxx	Copy Tape	Copies one or more consecutive programs from the input tape onto the output tape. If xxxxxx is specified, all programs up to but not including it are copied onto the output tape. When the program name xxxxxx is encountered, the tape is back-spaced over the record containing the name before control is returned to the user. If xxxxxx is not used, only one program is copied if a file mark is encountered before the program name is found, and the file mark message is displayed.
LI	List Input	Lists on the output comment device the names of all programs within a file on the input tape and stops at the first file mark. The file is searched until a NAM or EOL block is encountered. If a NAM block is found, the program name is displayed on the console. If an EOL block is found, *T is printed on the console. All NAM and EOL blocks within the input file are printed.
LO	List Output	Lists on the console the names of all programs within a file on the output tape. The functions are identical to those of LI.
TL	Terminate Load	Writes an EOL (end of load) block on the output tape. An EOL block consists of *T.
FM nn	File Mark	Writes a file mark on the tape associated with logical unit nn. The file mark is written where the tape is positioned.
CO	Close Output	Writes a file mark on the output tape, then rewinds and unloads the tape. The file mark is written where the output tape is positioned. Control is returned to the user when the request has been initiated. Thus, the user may continue operation while the tape is being rewound and unloaded.
RI	Rewind Input	Rewinds and unloads the input tape. Since control is returned to the user when the request is initiated, operation may be continued while the tape is rewinding.
RO	Rewind Output	Rewinds and unloads the output tape. The functions are identical to those of RI.
EX	Exit to RTOS	Causes an exit from the SMART program and returns control to RTOS.

† ∧ indicates a blank

Characters	Description
15 to 30	Address Field
32 to 72	Comment Field
75 to 80	Sequence Field

SETUP arranges each free-form statement into the above format and assigns identification and sequence numbers if desired. The first three characters of the name specified in the NAM statement are used for identification.

SETUP CONTROL STATEMENTS

Control statements allow the user to assign devices, perform updates, and accomplish certain file control functions. A slash begins each statement and also separates portions of the statements, while a carriage return terminates statements.

An example of teletypewriter input is:

/=D/xxxnnnnn

Device Assignment Statements

These allow logical units to be assigned. An example of a master-in statement is:

/=MI/nn

where nn is a two-digit decimal logical unit number.

This assignment of the input master tape to logical unit nn must be made prior to an update and refers to the magnetic tape unit containing the program to be updated. This assignment is not required during the source program create phase. If this statement is not used, input is assigned to the standard binary input device.

The master-out statement is:

/=MO/nn

This assignment of output to logical unit nn must be made prior to beginning the create/update. During a create or update, this magnetic tape unit contains the new, updated program. If this statement is not used, output is assigned to the standard binary output device.

Update Statements

These statements refer to specific source records that are not used during a create operation. Records must be updated in ascending numerical order. A source record may be referenced by only one update statement.

To delete one statement, use the form:

/=D/xxxnnnnn/xxxnnnnn

Where:

xxx	is the identification number.
nnnnn	is the sequence number of the record to be updated.

Any source language statements that follow the delete statement are written on the master output until another control statement is found.

The insert statement is:

/=I/xxxnnnnn

The source language statements following this instruction are inserted after statement number xxxnnnnn.

The end update statement is:

/=E

SETUP continues to copy and resequence the current program from input master to output. When a new NAM statement or a file mark is found on the input master, the input tape is positioned to the preceding record gap. Since more than one program can be updated without a full restart of SETUP, this statement is required to correctly end each update but does not terminate all operations.

File Control Statements

These statements allow the user to copy source programs without updating, to position to a particular program, to close files, and to exit to RTOS.

The copy control statement is:

/=CP/xxxxxx

It instructs the output master to copy the input master until program name xxxxxx is found and to halt at the record gap preceding the program NAM statement.

If no program name is specified, the entire input tape up to a file mark is copied, and the tape is positioned at the record gap preceding the file mark.

The copy process resequences the output tape unless the No Sequence option is selected. If multiple programs are copied, the identification sequence is restarted for each program.

The position input/output master tape format is:

/=PI/xxxxxx or /=PO/xxxxxx

These statements position the master tape to the program name xxxxxx and halt at the record gap preceding the program NAM statement. This function positions only in the forward direction.

If no program name is specified, the assigned tape is positioned at the record gap preceding the first file mark on the tape.

The close input master tape (CI) or output master tape (CO) format is:

/=CI or /=CO

With CI, rewind tape to load point; write a file mark and rewind the tape to load point.

The list instruction is:

/=LI or /=LO

It instructs the input master (LI) or output master (LO) to list the NAM records on the standard list device. These records are listed from the current position up to a file mark, and the tape is positioned at the record gap preceding the file mark.

SETUP enables the master-out file to be resequenced; the sequence statement format is:

/=S

SETUP can also inhibit resequencing the master-out file, thus maintaining the same identification and sequence numbers as the input master. Added or replaced statements are assigned blank sequence numbers. The no sequence statement format is:

/=NS

If neither /=S or /=NS is specified, sequencing is enabled.

The normal exit to RTOS causes SETUP operations to be terminated; it must be the last statement used. The format is:

/=EX

ERRORS

In addition to the standard RTOS system error comments, procedural errors may occur under SETUP. The following are typed out during operation of SETUP if the conditions are detected:

Message	Meaning
S01	The meaning and action taken depend on the control being processed. • Delete or insert – A file mark or end-of-tape reflective marker on the input tape was encountered before the specified statement was found. SETUP rewinds the input tape, writes a file mark on the output tape, and backspaces over it. • Copy and position – A file mark or end-of-tape reflective marker on the input tape was encountered before the specified program name was found. If processing a copy statement, the input tape is rewound to load point. If processing a position statement, the designated tape is rewound to load. • List – The end-of-tape reflective marker was detected before a file mark was encountered. The tape is rewound to load point.
S02	An illegal control statement was input.
S03	The end-of-tape reflective marker was sensed during an output to the master output tape. SETUP backspaces to the NAM statement for the program, writes a file mark, and backspaces over it.

To recover from the above errors, the teletypewriter bell is rung and a new control statement is accepted. The statement is processed according to its type and normal processing continues.

EXAMPLES

The following SETUP examples assume that logical unit 6 is on magnetic tape 0 and logical unit 7 is on magnetic tape 1.

Create PROGA via teletypewriter input.

Teletypewriter Input/Output	Action
SETUP	SETUP is in control.
/=MO/07	Assign magnetic tape 1 as output master.
NAM PROGA	
START ENA $13 LOAD A	Format into fixed-form records. Sequence beginning with PRO00001. Write on output master (tape unit 1).
END START	
/=CO	Write file mark on unit 1. Rewind tape.
/=EX	Exit to the RTOS System.

Using a tape containing two source programs (PROGA and PROGB) followed by a file mark, reverse their order and update PROGB.

Teletypewriter Input/Output	Action
SETUP	SETUP is in control.
/=MI/06	Assign unit 0 as input master.

Teletypewriter Input/Output	Action
/=MO/07	Assign unit 1 as output master.
/=PI/PROGB	Position input to second program.
/=D/PRO00003	Delete third record of PROGB.
/=E	Complete transfer of records. (Unit 1 now contains updated PROGB as the first source program.)
/=CI	Rewind unit 0.
/=CP/PROGB	Copy PROGA to unit 1 as second program.
/=CI	Close unit 0.
/=CO	Close unit 1.
/=EX	Exit to the RTOS system.

MAGNETIC TAPE UTILITY PROCESSOR (MTUP)

The magnetic tape utility processor (MTUP) is a general-purpose magnetic tape utility software product that may be used with RTOS or MSOS operating systems.

The magnetic tape utility processor is designed to provide a set of functional operations to process magnetic tape files created on CDC or other standard equipment. The capabilities of the magnetic tape utility processor encompass the more complex record formats and labeling structures normally found in business data processing environments. It is designed to:

- Provide standard ANSI magnetic tape labels.
- Provide a medium through which users may utilize a CYBER 18 system to reduce the burden of input/output processing on other systems, such as off-line printing of listable tapes, initializing of tapes, etc.
- Provide a medium for data file manipulation of CYBER 18 system-created tapes to augment other CYBER 18 features: readable and understandable tape dumps, blocking and unblocking of data files to improve input/output efficiency, etc.

CONFIGURATION REQUIREMENTS

The magnetic tape utility processor runs under control of RTOS with a minimum of 4K of available storage in the job area. It is installed on the system library and loaded into the job area by pressing the manual interrupt and entering:

```
MTUP
```

A conversational display terminal and two magnetic tape transports are required. Tape to print functions require only one transport. A printer is needed if the print and dump functions are to be used extensively.

FEATURES

The capabilities of the magnetic tape utility processor include:

- Dump – Print the tape in either hexadecimal or character mode.
- Print – Print the standard listing from tape.
- Copy – Copy tape to tape.
- Verify – Verify data for equality on two tape files.
- Initialize – Write volume 1 tape headers with volume serial numbers on any tape.
- Labels – Provision for writing and reading standard tape labels and trailer records.
- Blocking – All functions of the package can process blocked or unblocked records on either input or output.
- Record Formats – All functions of the system utility processor can process the following record formats.

  -Variable length unblocked records

  -Variable-length blocked records

  -Fixed-length records

  -Fixed-length blocked records

  -Undefined records; i.e., fixed- or variable-length records that follow no standard or are intermixed.

- Positioning – The capability to position tapes to given records, blocks, or records within a block using any of the processing functions.
- Conversion – All functions can select any of the following data conversion options:

  -ASCII to EBCDIC

  -EBCDIC to ASCII

  -ASCII to BCD

  -BCD to ASCII

  -EBCDIC to BCD

  -BCD to EBCDIC

- Selection – Select records for processing, based on specified criteria (optional module).

For further information refer to the Magnetic Tape Utility Processor Reference Manual.

UTILITY ASSEMBLER

The RTOS utility assembler provides the capability of generating relocatable binary object programs from source programs written in CYBER 18 assembly language. This language is the same as 1700 Series macro assembly language except that the following are not allowed:

- Macro instructions (MAC, EMC)
- Variable field definitions (VFD)
- Scaled decimal constants (DEC)
- Conditional statements (IFA, IFC, EIF)
- Use of special characters to terminate alpha strings (ALF . , msg.)

For detailed information refer to the CYBER 18 RTOS Assembler Reference Manual or the 1700 MSOS Macro Assembler Reference Manual.

ASSEMBLING SOURCE PROGRAMS

Load the assembler using the *L and *X statements or by entering the file name ASSEM. The assembler is loaded in the job area and must be absolutized at the location where it is to execute. When the assembler has been read into core, it types OPT and waits for input of options.

Set up the source programs on the input and enter the options desired. The following options may be input in any order:

Control Character	Usage
L	The program listing is written onto the standard list device during pass 2 (the first 72 characters of the list record).
D	When used with the L parameter, the full list record is output: 72 characters on the first record and the remaining characters on the second (split line printout).
F	When used with the L parameter, the entire list record is written on one line.
P	The relocatable binary program is written on magnetic tape.
M	Control is returned to the system.
C	Allows programs to be continuously assembled without operator intervention. Omission of this parameter causes the assembler to accept a new set of parameters after each assembly.
R	Rewinds all tapes and returns control to the system.
E	Causes an end-of-load (EOL) block to be written on binary output tape after each assembly. Omission of this parameter causes a default to a file mark on the binary output. In either case, the tape is backspaced over the record.
cr	A carriage return is the end-of-message indicator.

The assembler then reads the source input (pass 1). The name of the source program is typed on the comment device at the start of pass 1. Source input is then read up to the END statement, and pass 1 errors are typed out.

The input is then backspaced to the NAM card. (With card input the operator must replace the source deck in the reader.)

The assembler then rereads the source input (pass 2). List output is produced if requested, and relocatable binary output is punched if requested. Pass 2 errors are typed out. The assembly is completed when the END statement is reread.

If the C option was selected, the next program is assembled from the source input. This continues until either a MON card or end-of-file is read; control then returns to RTOS via the exit request.

If the C option was not selected, the assembler types OPT and requests input of new options before assembling the next program.

When the assembler encounters either a MON statement or a file mark, the message EOL REC is output to the console (if the P option was specified). Entry of cr or any entry other than *Y cr causes the output tape to be backspaced so that subsequent output writes over the end-of-load record (EOL block or file mark, see option E).

SYSTEM INITIALIZATION

System initialization is the procedure that loads the Real-Time Operating System (RTOS) programs, data, and other resident routines into main (core) memory and activates the monitor. Details of system initialization procedures are provided in the CYBER 18-10 and 18-20 RTOS Installation Handbook.

SYSTEM INITIALIZER

The system initializer loads the system programs into core from relocatable binary cards or magnetic tape. Initialization begins at the lowest location (towards 0) of core. The system initializer is provided as an absolute record on cards or magnetic tape and is loaded into the upper locations of available memory by a bootstrap loader.

The system initializer is composed of the following modules: a loader module (RCMLDR), a punch module (RCMPDR), a control module (RCMCON), and appropriate input drivers (same as the MSOS system initializer).

If it is necessary to generate a new system initializer by absolutizing these modules using the relocatable loader, then the control module must be the first module read in and the rest of the modules are loaded in higher core locations. All modules are in run-anywhere form.

CONTROL STATEMENTS

The system initializer accepts the following control statements.

*

This statement causes the system initializer to read the next control statement from the INPUT device. This causes initialization to begin if it is entered at the beginning of the program, or to continue if it is typed after a pause.

*S

This statement assigns a hexadecimal value to an entry name in the initializer loader table. There are two possible formats to this statement:

*S,name,hhhh

*S,name,P

Where:

name is an entry name.

hhhh is the hexadecimal value being assigned to it.

P assigns the current value of the location counter plus one to the name.

*L,hhhh

This statement begins (or continues) loading programs from the input device and absolutizing them in memory, beginning at location hhhh. If hhhh is not specified, the current value of the location counter (equal to 0 initially) is used. If hhhh is omitted, the comma is also omitted.

*F

This statement is used to advance the output tape forward one file. It may only be used in response to the READY OUTPUT message when it is desired to skip past a copy of the system loader (that has been copied to the output tape) and write the system file as the second file on the tape.

*G

This statement is used to advance the input tape forward one file. It may only be used in response to the READY OUTPUT message and should be used only for the two-pass mode with tape input to skip over the system initializer program (file 1) after the input tape has been rewound for pass 2.

*D

When this statement precedes the *L statement, the loader information concerning the previous data block declaration is deleted.

*T

This statement signals the end of the installation material to the initializer.

*I,hhhh

This statement allows the converter-equipment code of the initializer input device (card reader or tape unit) to be specified as any four-digit hexadecimal value.

*O,hhhh

This statement allows the converter-equipment code of the initializer output device (tape only) to be specified as any four-digit hexadecimal value.

The initializer is waiting for a control statement from the console whenever it has displayed the letter Q. It is also waiting for a command when the program is first begun and it has displayed SI.

SYSTEM INITIALIZER PROCEDURE

ONE-PASS INITIALIZING

Systems that include a job area and/or allocatable core area that is sufficient to accommodate the system initializer program and tables can be loaded in one pass.

In this mode of operation, programs are loaded directly into core and the load map and entry point symbol table are generated as the binary input is read in. When the end of load statement (*T) is read, the message:

    READY OUTPUT

is typed if an output magnetic tape unit is available. This unit can be the same as the unit used to read the binary input. Provision is made to optionally skip one file (*F) prior to output of the absolutized system. The resulting system file can be read back in by the system loader if core is destroyed.

TWO-PASS INITIALIZING

If the applications program system is too large for available core after loading the initializer, the resident programs must be initialized in two passes, provided that a magnetic tape unit is available for punch output (separate from the input unit).

In this mode of operation the binary input is read once to build an entry point symbol table and produce the load map. The message:

    END PASS 1 – READY OUTPUT

is then typed. The input is reread and each binary program is loaded, absolutized, and then output to tape before loading the next program into the same available core. Finally, the absolute programs are read into core from the output tape by the system loader.

The two-pass method avoids the entry point linkage restrictions that were originally imposed by the two-part system initializer.

SYSTEM LOADER

A system loader is provided to load the absolutized system from magnetic tape. Installation details for the system loader are found in the CYBER 18-10 and 18-20 RTOS Installation Handbook.

The RTOS system may be initialized on a host CYBER 18 or 1700 computer system and the absolutized magnetic tape output may then be loaded on the actual CYBER system. The RTOS system initializer does not provide for punching absolute binary cards directly. However, the system tape produced by the initializer may be converted to cards and the cards may then be loaded.

The system loader is loaded by the same (manually entered) bootstrap as the system initializer. Typically, the loader is set up as file 0 and the absolutized system as file 1 of the system library tape.

ENGINEERING FILE

The engineering file preserves driver error information for system maintenance. To save memory, RTOS utilizes available space in the physical device tables.

DEVICE FAILURE HANDLING

When an input/output driver determines that an error condition has occurred, it reports the error to the error logging routine in RCMEFD. The re-entrant calling sequence is:

	15 6	5 0
Q register:	LOGICAL UNIT	ERROR CODE

I register: Set to the driver physical device table address.

The format to call the error logging routine is:

```
RTJ+    LOG.
```

An alternate version RCMEFM is used instead of RCMEFD for systems with mass storage devices. For mass memory failures, RCMEFM also logs the failure on the system comment output device with the message:

```
MM ERR xx LU=yy S=ssss
```

Where:

xx	is the error code.
yy	is the logical unit.
ssss	is the hardware status.

DEVICE FAILURE STORAGE

Words 13 and 14 of the physical device table are used by RTOS to save the number of errors and the status after the last error. In MSOS these words are used to designate the mass-memory sector number and length for a mass-memory resident driver, respectively.

Physical Device Table

	15 10	9 0
WORD 13	ecode	ERROR COUNT
WORD 14	STATUS ON LAST ERROR	

If ecode = $3F_{16}$, then all errors are counted. The error count is the number of errors that have occurred having a code equal to the error code specified by ecode. When any error occurs the last status is transferred from word 12 to word 14.

DEVICE FAILURE LISTING (EFLIST)

A nonresident utility routine is provided that may be loaded and executed under RTOS to print the engineering file. It is loaded from the input unit using the *L statement or from the system library by entering the file name EFLIST. After loading and initial setup, the job processor types J.

To list the engineering file for logical unit lu without resetting the error count, enter:

```
*O,lu
```

If lu is blank, all units are listed.

To set the error code to ec and reset the error count to 0 for the specified lu, enter:

```
*O,lu,ec
```

ec is a decimal number from 0 to 62. If ec is greater than or equal to 63, all errors are counted.

Engineering file data is typed in the following format:

```
LOGICAL UNIT nn description

STATUS ON LAST ERROR ssss

CURRENT STATUS cccc

nnnn ERRORS OF CODE ec
```

Where:

nn	is the logical unit number.
description	is obtained by decoding the physical device table type code.
ssss	is the status on the last error (of any type).
cccc	is the current status (word 12).
nnnn	is the number of errors that have been counted.
ec	is the error code that was counted. If ALL, all errors were counted.

On completion, J is typed.

GLOSSARY

The glossary is intended to assist in the communication of facts and ideas related to information processing.

In all instances, a comparison has been made to the American National Standards Institute (ANSI) glossary to ensure consistency with standard nomenclature wherever possible.

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 PROGRAM – A program composed of command sequence storage information, which may be loaded by a checksum loader

ADC – Analog-to-digital converter

AGENCY – A composition of processors dedicated to performing a single task

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.

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

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)

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.

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.

BUFFERING – Overlapping execution of one or more I/O routines with the execution of the program that called them

BYTE – A sequence of adjacent binary digits operated upon as a unit and usually shorter than a word; within the CYBER 18, a byte is eight bits.

CALIBRATION – Conversion of a quantity into measurable units (engineering units)

CENTRAL MEMORY – Refers to the directly addressable core storage of computers; abbreviated as CM

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.

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 signal by opening a closed electrical connection; abbreviated as CC

CONTROLLER – A hardware device that controls access and data transfer to I/O units which are connected to it

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 CM; abbreviated as CMR.

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.

DAC – Digital-to-analog converter

DAISY CHAIN – Refers to a hardware capability to connect devices in series up to a maximum channel length

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

DATA REDUCTION – The process of transforming data into intelligible form by averaging, smoothing, adjusting, scaling, and ordering experimental readings

DDC – Direct digital control

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 ROUTINE – A program or routine designed 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; abbreviated as DAC

DIRECT DIGITAL CONTROL – A closed loop control system in which the output depends directly on input and computation (all in one frame time); abbreviated as DDC

DIRECT STORAGE ACCESS – Method of accessing blocks of data directly in CYBER 18 core memory by the peripheral equipment, without using the A/Q channel; abbreviated as DSA

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

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

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

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

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.

FIELD LENGTH – The number of central memory words that a program occupies; abbreviated as FL

FILE ORDINAL – A number equated to a mass storage file for the duration of the job

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.

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

IFIPS – International 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.

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

JOB TERMINATION – Those activities necessary to logically terminate job execution

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 open if power fails); in connection with contact closures, the term 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 is done to the external devices in the event of a power failure.

LOADING – The process of transferring a program from external devices to storage; in RTOS the relocatable loader transfers a relocatable program to the first sequential available positions in core.

LOGICAL UNIT – A number that identifies a specific I/O device or function

MAN-MACHINE COMMUNICATION – Software components that establish communication between the operating system and the operators

MASS-STORAGE RESIDENT – That 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

MEMORY PROTECT – Hardware and software that prevent batch programs from destroying priority or operating system core storage

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 for an open-ended device and R -1 for a closed (end-around) device, where R 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 use twos complement arithmetic; devices with modulus 2 -1 use ones complement.

NONREUSABLE PROCEDURE – A procedure that is destructive and noninitializing.

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 1.

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

OVERLAY PROCESSING – A technique for processing a program whose total storage requirement for instructions exceeds available memory; the user divides the program into elements that 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.

PART 0 – A user-defined block of contiguous memory extending from location 0 up to the location ENDOV4; Part 0 must be less than 32,768 words.

PART 1 – Part 1 is the block of contiguous memory immediately following part 0 and extending to the highest available core location, 65K (for MSOS only).

PARTITION – One of a number of segments of a given area in core into which a mass-storage resident program may be read and executed

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

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.

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

PSR – Programming systems report

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 has its own data storage area and so that all valuable information is stored if the processor is interrupted.

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.

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

RUN-ANYWHERE – Programs that execute properly regardless of where they are executed in core memory; all data internal to the program is referenced by relative addressing.

SAMPLING RATE – The rate at which collections of physical quantities are made; e.g., if a pressure is measured each microsecond, the sampling rate is 1000 measurements per second or 1000 Hz (1 kHz)

SCALING – Changing the value of a quantity by a factor in order to bring its range within prescribed limits

SECTOR MARKS – Disk tracks that are marked off in equidistant points for addressing purposes

SEQUENTIAL FILE – A file organized so that records are processed one after the other in their physical sequence; the records may or may not be in a logical sequence. Access to the records is in a serial manner.

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

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

SPOOLING – A technique of transferring jobs and data from one input device to another (usually a mass storage file) for processing later

STACK – A stack may be either a pushdown or pop-up stack. In a pushdown, pop-up stack all entries must be contiguous.

SWAP AREA – A predefined area on mass memory where jobs may be swapped

SWAPPING – The transfer of a program in unprotected central memory to mass storage, making the area available to the protected foreground

SYNCHRONOUS – Pertaining to a system in which all operations and events are controlled by equally spaced pulses from a clock

THREAD – A list of entries that each contain a pointer to the next entry; e.g., logical unit thread

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 ACCURACY – A parameter defining the imposed tolerance on intervals between point acquisition. Signals with scan frequencies $\leq$ 800 Hz to be 0.1% time accurate; i.e., at f scan = 100 Hz, each time interval (TI) should fall within the following limits:

10 milliseconds - 10 microseconds $\leq$ TI $\leq$ 10 milliseconds + 10 microseconds (or 9.99 milliseconds $\leq$ TI $\leq$ 10 milliseconds)

TRANSDUCER – A device for converting energy from one form to another

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 PROGRAM – An object program loaded and entered under RTOS control; includes batch and priority programs and library routines

USER'S EXECUTABLE CODE – That portion of a program that represents steps that the computer performs; 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.

WORST CASE – That which gives maximum stress or consumes maximum time; e.g., the pattern of 0s and 1s in storage that creates the greatest noise or the maximum possible time between two significant programming operations

SYSTEM DIRECTORY B

RTOS allows the user to add a customized system directory that consists of a table of absolute program starting addresses. Other programs may use the standard scheduler request and specify an index to the system directory instead of an actual address (refer to section 3).

The directory consists of one-word entries, with the first entry having the entry point name RAT (routine address table). The address of RAT is stored in location EB_{16} in low core. Figure B-1 is an example of a system directory.

The SYSDAT program must be re-assembled and installed with the system resident programs.

RTOS does not use internal system directory calls.

```
*
*        USER-SUPPLIED SYSTEM DIRECTORY ADDED TO SYSDAT
*
               EQU      RAT(*),SLDIRY(*)
               ADC      PROG0             SYSTEM DIRECTORY INDEX 0
               ADC      PROG1             SYSTEM DIRECTORY INDEX 1
               ADC      PROG 2            SYSTEM DIRECTORY INDEX 2

               EXT      PROG0
               EXT      PROG1
               EXT      PROG2

                                          END OF SYSDAT
```

Figure B-1. System Directory Example

COMMUNICATIONS REGION

The area of core from 0 to FF is used as a communications area because it can be addressed directly by a one-word instruction. Its contents are defined in table C-1; all locations are protected except as noted.

Options are provided to reduce core requirements by loading RTOS stacks into low-core areas 47 through B2 and/or C3 through E7. Refer to the RTOS Installation Manual.

MSOS uses the following locations for different purposes than RTOS: C0, C1, E7, EE, and F3. The MSOS extended core table is not required by RTOS. Refer to the MSOS Reference Manual.

TABLE C-1. COMMUNICATIONS AREA CONTENTS

Location	Contents	Hexadecimal/ Equivalent
0	0001010000000000	1400
1	Address of system restart routine	
2	0000000000000000	0
3	0 01	1
4	0 011	3
5	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	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	0 0	0000
23	0 1	1
24	0 10	2
25	0 100	4
26	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
2C	0 10 0	200

TABLE C-1. COMMUNICATIONS AREA CONTENTS (Continued)

Location	Contents	Hexadecimal/ Equivalent
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	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	0 101	5
44	0 110	6
45	0 1001	9
46	0 1010	A
47-B2	Reserved for user applications	
B3	Logical unit number of scratch unit	
B4	Top of thread of entries in schedule stack	
B5	Location of FNR	
B6	Address of complete request subroutine used by drivers	
B7	Address of MASKT	
B8	Core location of top of interrupt stack	
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	RTOS binary to ASCII decimal subroutine (unprotected)	
C1	RTOS binary to ASCII hexadecimal subroutine (unprotected)	
C2	Logical unit number of the library unit	
C3†	Most significant sector number of first program library directory block	
C4†	Least significant sector number of first program library directory block	
C5-E3	Reserved for FORTRAN (unprotected)	
E4	Used for load-and-go (unprotected)	
E5	Reserved for FORTRAN (unprotected)	
E6†	Length of system library directory	
E7	Address of LOG1A logical unit table	
E8	Real-time clock counter	
E9†	Core address of extended core table	
EA	Location of dispatcher	
EB	Core location of system library directory	
EC	Temporary highest unprotected location + 1	
ED	Temporary lowest unprotected location - 1	
EE	RTOS ASCII field decode subroutine (unprotected)	
F0	Core location of first available volatile storage	
F1	Length of table of preset entry points	

†Used by MSOS; always 0 in RTOS.

TABLE C-1. COMMUNICATIONS AREA CONTENTS (Continued)

Location	Contents	Hexadecimal/ Equivalent
F2	Location of table of preset entry points	
F3	RTOS MI subprogram set-up and error routine (unprotected)	
F4	Location of entry for system requests	
F5	Largest core location used	
F6	Highest unprotected location +1 (end of job area +1)	
F7	Lowest unprotected location -1 (start of job area -1)	
F8	Address of internal interrupt processor	
F9	Logical unit number of standard input device	
FA	Logical unit number of standard binary output device	
FB	Logical unit number of standard print output device	
FC	Logical unit number of output comment device	
FD	Logical unit number of input comment device	
FE	Location of interrupt stacker program	
FF	Memory index register I (unprotected)	

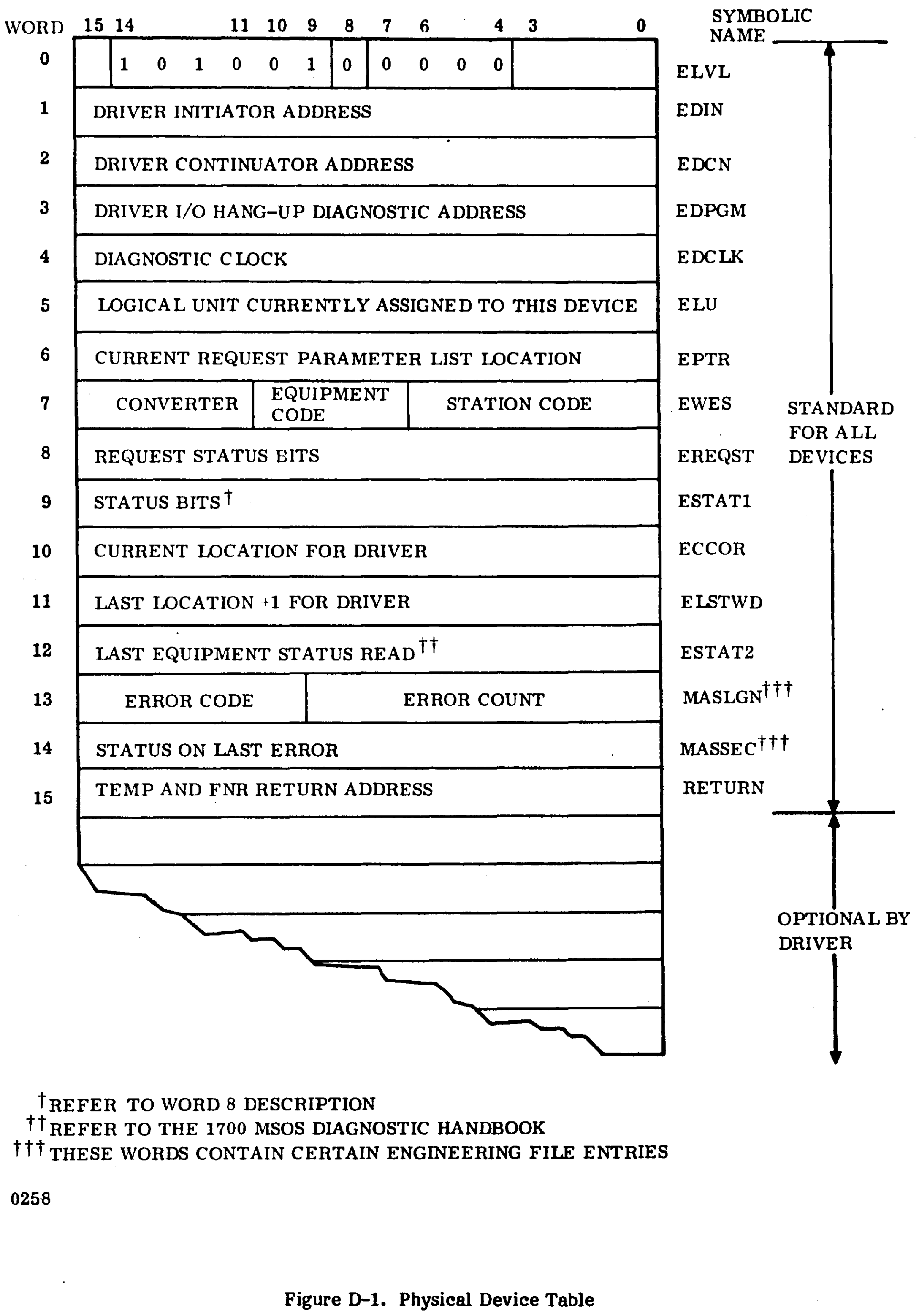
PHYSICAL DEVICE TABLE

The physical device tables are included in SYSDAT (the system and parameters program).

RTOS utilizes words 13 and 14 (MASLGN and MASSEC) for engineering file storage (drivers are not allowed to be mass-storage resident under RTOS); otherwise, the definition of these tables is identical to the MSOS definition.

PHYSICAL DEVICE TABLE

Each device has a physical equipment table that contains the interfacing information specified by the user to the device (figure D-1). 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 that 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 special use by drivers.



WORD	15 14 11 10 9 8 7 6 4 3 0	SYMBOLIC NAME	
0	| 1 0 1 0 0 1 | 0 | 0 0 0 0 |	ELVL	STANDARD FOR ALL DEVICES
1	DRIVER INITIATOR ADDRESS	EDIN	
2	DRIVER CONTINUATOR ADDRESS	EDCN	
3	DRIVER I/O HANG-UP DIAGNOSTIC ADDRESS	EDPGM	
4	DIAGNOSTIC CLOCK	EDCLK	
5	LOGICAL UNIT CURRENTLY ASSIGNED TO THIS DEVICE	ELU	
6	CURRENT REQUEST PARAMETER LIST LOCATION	EPTR	
7	CONVERTER | EQUIPMENT CODE | STATION CODE	EWES	
8	REQUEST STATUS BITS	EREQST	
9	STATUS BITS†	ESTAT1	
10	CURRENT LOCATION FOR DRIVER	ECCOR	
11	LAST LOCATION +1 FOR DRIVER	ELSTWD	
12	LAST EQUIPMENT STATUS READ††	ESTAT2	
13	ERROR CODE | ERROR COUNT	MASLGN†††	
14	STATUS ON LAST ERROR	MASSEC†††	
15	TEMP AND FNR RETURN ADDRESS	RETURN	
			OPTIONAL BY DRIVER

†REFER TO WORD 8 DESCRIPTION
††REFER TO THE 1700 MSOS DIAGNOSTIC HANDBOOK
†††THESE WORDS CONTAIN CERTAIN ENGINEERING FILE ENTRIES

0258

Figure D-1. Physical Device Table

ASCII CONVERSION TABLE

The 1963 American Standard Code for Information Interchange (ASCII) is used by RTOS. ASCII code uses eight bits: bit 8, which is always 0, 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.

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	Device controls
DC_3	001 0011	13	Device controls
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	Information separators
S_2	001 1010	1A	Information separators
S_3	001 1011	1B	Information separators
S_4	001 1100	1C	Information separators
S_5	001 1101	1D	Information separators
S_6	001 1110	1E	Information separators
S_7	001 1111	1F	Information separators

8-Bit ASCII Codes	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape	8-Bit ASCII Codes	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape
20_{16}	No Punch	No Punch	20_8	40_{16}	0-8-7	8-4	37_8
21†	11-8-2	12-8-7	52	41	12-1	12-1	61
22	8-7	8-7	17	42	12-2	12-2	62
23†	12-8-7	8-3	77	43	12-3	12-3	63
24	11-8-3	11-8-3	53	44	12-4	12-4	64
25†	0-8-5	0-8-4	35	45	12-5	12-5	65
26†	8-2	12	00 (35)††	46	12-6	12-6	66
27†	8-4	8-5	14	47	12-7	12-7	67
28	0-8-4	12-8-5	34	48	12-8	12-8	70
29†	12-8-4	11-8-5	74	49	12-9	12-9	71
2A	11-8-4	11-8-4	54	4A	11-1	11-1	41
2B†	12	12-8-6	60	4B	11-2	11-2	42
2C	0-8-3	0-8-3	33	4C	11-3	11-3	43
2D	11	11	40	4D	11-4	11-4	44
2E	12-8-3	12-8-3	73	4E	11-5	11-5	45
2F	0-1	0-1	21	4F	11-6	11-6	46
30	0	0	12	50	11-7	11-7	47
31	1	1	01	51	11-8	11-8	50
32	2	2	02	52	11-9	11-9	51
33	3	3	03	53	0-2	0-2	22
34	4	4	04	54	0-3	0-3	23
35	5	5	05	55	0-4	0-4	24

†Refer to note 1 below.
††Refer to note 3 below.

NOTES

1. 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.
2. The CDC Standard 1.10.003 is supported by an assembly option. For CDC ASCII mode of operation, the card punches 12-8-2 and 12-0 are stored internally as 7B. The card punches 11-8-2 and 11-0 are stored internally as 7D. For line printer operations, the internal codes 7B and 7D are converted to 5B and 5D to allow printing the hardware compatible graphic characters [(left bracket) and] (right bracket).
3. Since 1832 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 35_8.

   On tape read operations, the BCD code 35_8 is always translated to an ASCII $25 (%).

8-Bit ASCII Codes	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape	8-Bit ASCII Codes	026 Punches	029 Punches	6-Bit Extended BCD Magnetic Tape
36	6	6	06	56	0-5	0-5	25
37	7	7	07	57	0-6	0-6	26
38	8	8	10	58	0-7	0-7	27
39	9	9	11	59	0-8	0-8	28
3A	8-5	8-2	15	5A	0-9	0-9	31
3B	11-8-6	11-8-6	56	5B†	12-8-5	12-8-2	75
3C†	12-8-6	12-8-4	76	5C†	0-8-2	0-8-2	36
3D†	8-3	8-6	13	5D†	11-8-5	11-8-2	55
3E†	8-6	0-8-6	16	5E	11-8-7	11-8-7	57
3F†	12-8-2	0-8-7	72	5F†	0-8-6	0-8-5	32

†Refer to note 1 below.

NOTES

1. 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.
2. The CDC Standard 1.10.003 is supported by an assembly option. For CDC ASCII mode of operation, the card punches 12-8-2 and 12-0 are stored internally as 7B. The card punches 11-8-2 and 11-0 are stored internally as 7D. For line printer operations, the internal codes 7B and 7D are converted to 5B and 5D to allow printing the hardware compatible graphic characters [(left bracket) and] (right bracket).
3. Since 1832 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 35_8.

   On tape read operations, the BCD code 35_8 is always translated to an ASCII $25 (%).

ASCII to EBCDIC†		ASCII to EBCDIC		ASCII to EBCDIC	
0 to 1F††	3E	3F	6F	5F	5F
20	40	40	3F	60	79
21	5A	41	C1	61	81
22	7F	42	C2	62	82
23	7B	43	C3	63	83
24	5B	44	C4	64	84
25	6C	45	C5	65	85
26	50	46	C6	66	86
27	7D	47	C7	67	87
28	4D	48	C8	68	88
29	5D	49	C9	69	89
2A	5C	4A	D1	6A	91
2B	4E	4B	D2	6B	92
2C	6B	4C	D3	6C	93
2D	60	4D	D4	6D	94
2E	4B	4E	D5	6E	95
2F	61	4F	D6	6F	96
30	F0	50	D7	70	97
31	F1	51	D8	71	98
32	F2	52	D9	72	99
33	F3	53	E2	73	A2
34	F4	54	E3	74	A3
35	F5	55	E4	75	A4
36	F6	56	E5	76	A5
37	F7	57	E6	77	A6
38	F8	58	E7	78	A7
39	F9	59	E8	79	A8
3A	7A	5A	E9	7A	A9
3B	5E	5B	4A	7B	C0
3C	CE	5C	FA	7C	6A
3D	7E	5D	CC	7D	D0
3E	EC	5E	6D	FF	A1
				7F	3E

†All codes are in hexadecimal notation.

††Invalid ASCII code; conversion is made to invalid EBCDIC code (3E).

EBCDIC to ASCII†		EBCDIC to ASCII		EBCDIC to ASCII	
0 to 2F††	7F	83	63	CA, CB	7F
3F	40	84	64	CC	5D
40	20	85	65	CD	7F
41 to 49††	7F	86	66	CE	3C
4A	5B	87	67	CF	7F
4B	2E	88	68	D0	7D
4C	7F	89	69	D1	4A
4D	28	8A to 90††	7F	D2	4B
4E	2B	91	6A	D3	4C
4F	7F	92	6B	D4	4D
50	26	93	6C	D5	4E
51 to 59††	7F	94	6D	D6	4F
5A	21	95	6E	D7	50
5B	24	96	6F	D8	51
5C	2A	97	70	D9	52
5D	29	98	71	DA to E1††	7F
5E	3B	99	72	E2	53
5F	5F	9A to A0††	7F	E3	54
60	2D	A1	7E	E4	55
61	2F	A2	73	E5	56
62 to 69††	7F	A3	74	E6	57
6A	7C	A4	75	E7	58
6B	2C	A5	76	E8	59
6C	25	A6	77	EA, EB††	7F
6D	5E	A7	78	EC	3E
6E††	7F	A8	79	ED, EF††	7F
6F	3F	A9	7A	F0	30
70 to 78††	7F	AA to BF††	7F	F1	31
79	60	C0	7B	F2	32
7A	3A	C1	41	F3	33
7B	23	C2	42	F4	34
7C	7F	C3	43	F5	35
7D	27	C4	44	F6	36
7E	3D	C5	45	F7	37
7F	22	C6	46	F8	38
80	7F	C7	47	F9	39
81	61	C8	48	FA	5C
82	62	C9	49	FB to FF	7F

†All codes are in hexadecimal notation.

††Invalid EBCDIC code or no equivalent ASCII code; conversion is made to invalid ASCII code (7F).

ERROR CODES AND MESSAGES F

The error codes and messages listed in tables F-1 through F-16 appear on the conversational display terminal when the conditions they describe occur.

TABLE F-1. SYSTEM INITIALIZER ERRORS

Error Code	Description
E1	Checksum error on input device
E2	Incorrect *L,hhhh control statement. hhhh is below the last storage location or in the common storage area.
E3	Incorrect or out-of-order binary record
E4	Incorrect common or data storage reservation
E5	Memory overflow into symbol table; last program loaded is too long
E6	Attempt to load program below top of existing programs
E7	Data storage assigned beyond storage limit
E8	Duplicate entry point
E9	Insufficient core to absolutize a program during pass 2.
E10	Unpatched external(s)
E11	Illegal control statement
E12	Two programs reference the same external name, one with relative addressing and the other with absolute addressing.
E13	Entry name thread broken while linking
E14	External name thread broken while linking
SLEW	Irrecoverable error condition (E1 through E14). The program (program 1) preceding the program containing the error (program 2) is loaded; program 2 is not loaded; the program following program 2 writes over program 2.
Q	Control statement is requested from teletypewriter. Enter * cr to read the next control statement from input device.
SI	System initializer is ready to accept first control statement from the conversational display terminal. Enter * cr to read next control statement from input device.
END PASS1	Indicates completion of pass 1 if two-pass mode was used. Rewind the input tape and enter * cr or *G cr .
READY OUTPUT	System initializer is ready to punch output. Set up output tape and enter * cr or *F cr .

TABLE F-2. SYSTEM ERRORS

Message	Description
ACTION	Follows the LU,FAILED error message when device error routine is used or no alternate exists Respond to ACTION by typing one of the following: CD Inform any future programs calling the device of failure by passing Q register with bit 15 = 1 to their completion addresses. Error is reported to calling program and device is marked down. No subsequent attempt is made to operate this device. CU Report error to requesting program and continue processing requests. DD Activate CU and suspend job processing. DU Activate CU and suspend job processing. RP Repeat request.
ALT,lu	Follows ACTION error message when alternate device handler is used and an alternate exists. lu is the logical unit of alternate device.
LU,luFAILED xx ssss	I/O driver cannot recover from error. lu is the logical unit of failed device. xx is the code indicating cause of failure. ssss is the hardware status of failed device.
MI	Manual input processor ready to accept input from the conversational display terminal
OV	Overflow of volatile storage. System requires more volatile core than was provided in SYSDAT. System hangs.
PE	Parity error; system hangs.
PF	Protect fault; system hangs.
PP	Requests that PROGRAM PROTECT switch be set. Typed on system restart if protect processor is present
PW	Power failure; system hangs.
RTOS 3.0 xxx	System identification typed on system restart xxx is the system level SYSLVL (PSR summary level)

TABLE F-3. JOB PROCESSOR ERROR CODES

Message	Description
MI	Manual interrupt processor is ready to accept input from the conversational display terminal.
J	Job processor (manual interrupt processor) is ready to accept input from the conversational display terminal.
ER04	Invalid parameters in the job control statement
ER05	Invalid job control statement; processing subprogram is not present
ER09	I/O error while searching the system library, terminates search
ER10	Specified file name not found on system library
ER14	Program to be loaded exceeds the size of available core
ER20	Irrecoverable I/O error
NOTE: The job processor expects a control statement following the ERnn error message (i.e., J is implied).	

TABLE F-4. PROTECT PROCESSOR DIAGNOSTICS†

Diagnostic	Description
IR00,hexnum	Invalid request – The code 00 implies an unspecified error occurred in an unprotected request located at the address specified by hexnum.
IR01,hexnum	Invalid request code was specified in the request located at hexnum.
IR02,hexnum	Invalid request at the address specified by hexnum. The request processor specified by the request code is not contained in the system.
IR03,hexnum	Invalid buffer and length specified at the address hexnum. The sum of the first word address and length parameters of a read request extends beyond the top of unprotected memory.
IR04,hexnum	Invalid buffer address in a read request at hexnum. The request specified a buffer address starting in protected memory.
IR05,hexnum	Invalid buffer address specified in request located at hexnum. The buffer address is either negative or results in memory wraparound.
IR06,hexnum	Invalid length parameter in request located at hexnum. Length specification is either 0 (formatted only) or negative.
IR07,hexnum	Invalid completion address parameter in request at location hexnum. The completion address specified is either negative or protected.
IR08,hexnum	Invalid logical unit parameter in request located at hexnum. The logical unit parameter is incorrect or specifies a device driver that is not included in the system.
IR09,hexnum	Invalid motion control parameter(s) in request at location hexnum. P2, P3, or the density specification is greater than 7 or incorrect.
IR10,hexnum	Invalid priority parameters in request located at hexnum. Priority must be 0 or 1 for unprotected programs.

†These diagnostics are given if the protect processor is resident and the protect switch is ON when an unprotected program attempts to (illegally) jump or store into protected core.

TABLE F-5. LOADER ERROR DIAGNOSTICS

Code	Description
E01	Indicates an unrecoverable I/O error as a result of reading a block of input
E03	This is an unrecoverable error indicating an illegal input block or an input block out of order in the relocatable binary program.
E04	Indicates faulty common or data storage block reservation. During a load operation, the first program to declare common storage did not declare the largest amount. The loader continues to use the previously declared length.
E05	Indicates overflow of available memory during a load operation. This error occurs if the loader attempts to assign data storage to, or to load a program into, an area of memory above that available. The upper limit of available core is the lowest address of either the common storage block reservation or the loader's table.
E06	Results from attempting to load command sequence data, either in an RBD or a BZS block, below the lower limit of available memory. If processing an RBD block, the loader resumes operation by reading the next block of input. If processing a BZS block, the loader resumes operation by processing the next entry in the current block of input.
E7	Results from overflowing the data storage block reservation while loading command sequence data from either an RBD or a BZS block. If processing an RBD block, the loader resumes operation by reading the next block of input. If processing a BZS block, the loader resumes operation by processing the next entry in the current block of input.
E8	Indicates an illegal attempt to duplicate entry point names from several relocatable binary programs. Externals reference only the first occurrence of the entry point. Following error printout, the loader deletes any program attempting this illegal entry.
E10	Indicates an unlinked external (an external name in one program not matched by an entry point in any program of the relocatable binary input) when loading is complete. The loader also prints the name of the unlinked external and then resumes operation.
E12	Indicates that the loader encountered two programs referencing the same external name, one using absolute addressing and the other using relative addressing.
E13	The loader did not encounter a name for a transfer address during a loading procedure, or the name it did encounter is not defined as an entry point name in the loader's table.

TABLE F-6. SYSTEM LIBRARY EDITOR ERRORS†

Message	Description	Action
LIB01	Attempt to add a file header with a duplicate name of an existing file	Delete old file or use a different name and retry function.
LIB02	Reserved	Use longer tape or start new library tape.
LIB03	Reserved	List library and obtain correct names. Retry function.
LIB04	Reserved	Use a smaller Tape SCOPE system.
LIB05	Invalid command specified	Correct and retry command.
LIB06	Invalid file name specified in ADD, COPY, or GET command	Use a correct name and retry command.
LIB07	Invalid decimal or hexadecimal number was specified.	Correct parameter and retry command.
LIB08	Invalid priority specified in ADD command priority is greater than decimal 15.	Correct priority and retry command.
LIB09	Invalid parameter was specified.	Eliminate or correct parameter and retry function.
LIB10	An invalid date was specified in the ADD/ ADDP statement.	Correct the date and retry the add function.
LIB11	A record larger than the size of available memory has been read and truncated by the copy function.	The record cannot be copied.
LIB12	A parameter other than F, L, or T was specified in the SET command.	Correct parameter and retry function.
LIB13	An invalid hexadecimal number was specified in the SET command.	Correct number and retry function.
LIB14	An invalid logical unit number was specified in the *K command.	Correct logical unit number and retry function.
LIB15	The file specified in a GET command was not found.	Enter proper file name and retry function.
LIB16	The length of the program to be added was zero (or negative).	Load the program or use GET, LOADER and retry function.

†LIBEDT error messages describe the failure of a LIBEDT function.

TABLE F-7. BINARY TAPE EDITOR (SMART) ERRORS

Message	Description
??	Invalid control statement and/or parameters

TABLE F-8. SOURCE TAPE EDITOR (SETUP) ERRORS

Message	Description
S01	The meaning and action taken depend on the control being processed: Delete or Insert: A file mark or end-of-tape reflective marker on the input tape was encountered before the specified statement was found. SETUP rewinds the input tape, writes a file mark on the output tape, and backspaces over it. Copy and Position: A file mark or end-of-tape reflective marker on the input tape was encountered before the specified program name was found. If processing a copy statement, the input tape is rewound to load point; if processing a position statement, the designated tape is rewound to load point. List: End-of-tape reflective marker was detected before a file mark was encountered. The tape is rewound to load point.
S02	An illegal control statement was input.
S03	The end-of-tape reflective marker was sensed during an output to the master output tape. SETUP backspaces to the NAM statement for the program, writes a file mark, and backspaces over it.

TABLE F-9. MAGNETIC TAPE UTILITY PROCESSOR (MTUP) ACTION ERRORS†

Message	Description	Action
*DATA SET NAME:	Label processing – Output volume requires data set name if not available from input.	DSN="xxxxx"
*INVALID PARAM="xxx..." *RETYPE PARM:_____	Characters within quotes are invalid and may be corrected.	Enter corrected parameter.
*MOUNT, OUTPUT, SCRATCH	See initialize function (section 7).	ⓒⓡ implies tape is ready; any other character followed by cr implies terminate initialize.
*NEXT:	System has completed all prior requests and can accept next request.	Any utility operational or declarative statement

†These messages require operator response before continuing.

TABLE F-9. MAGNETIC TAPE UTILITY PROCESSOR (MTUP) ACTION ERRORS (Continued)

Message	Description	Action
10 ERRORS CONTINUE:	Verify function has located 10 consecutive records in errors.	Type (cr) to terminate; type one character followed by (cr) to continue.
*VOLSER=nnnnnn:	See initialize function (section 7).	See initialize function.
VOLSER=nnnnnn	Informative tape file just opened has the specified volume serial number.	None
VOL NOT EXPIRED USE:	Label processing – Output volume header records are checked against the system date.	(cr) implies do not use. U implies use ignoring expiration date.

TABLE F-10. MAGNETIC TAPE UTILITY PROCESSOR SYSTEM ERRORS†

Descriptive Errors(1)		
Message	Description	Action
FILE(S) NOT OPEN	Required file is not open and specified function cannot be executed.	Open file and re-enter function.
*FUNCTION NOT AVAILABLE	Attempted function is not available in system. Function is not invalid; rather the system was configured without the requested module.	Use another function if possible.
INCORRECT VOL MOUNT:	Volume mounted does not contain volume label, or header level sequence. is incorrect; i.e., with multivolume files, the wrong volume is mounted.	Mount correct volume and type carriage return.
*INVALID OPEN OR CLOSE	File being opened or closed is already open or closed.	Open or close proper file, or close and re-open file.
*PARM NOT AVAILABLE	Parameter is not available in system. Parameter is not valid; rather the system was configured without the requested module.	Use another parameter if possible.
Critical Errors(2)		
Code	Description	Action
****C000****	Data buffer link has been destroyed by I/O malfunction or CPU malfunction.	Reload utility.

1. Descriptive messages indicate the error and implicitly indicate that the previous function was not executed.
2. A critical message implies an error has occurred that prevents further utility processing.

†System error messages are always issued to the comment device.

TABLE F-10. MAGNETIC TAPE UTILITY PROCESSOR SYSTEM ERRORS (Continued)

Serious Errors(3)		
Code	Description	Action
****S000****	Available core has been filled.	Free core by closing file.
****S001****	Attempt to close file already closed.	Close proper file.
****S002****	1. Read end-of-file. 2. Attempt to write on file not opened for write. 3. I/O error; i.e., parity, read or write, lost data, or alarm error.	Retry function.
****S003****	Variable length block does not match actual length read, or variable read length is greater than specified block size.	Close all files. Open input as undefined and dump records to locate erroneous record. File cannot be processed as variable length.
****S004****	Blocking has been requested and specified block size is smaller than specified record size.	Re-open file with proper parameters.
****S005****	Variable size error detected prior to write.	Attempt to re-execute function after closing and re-opening all files. Possible hardware malfunction.
****S006****	Fixed blocked error detected prior to write. Record length is not specified.	Close file and re-open with proper record size or dump file to locate erroneous records.
****S007****	Labeled file sequence number in error. (File is not opened.)	Mount proper volume and re-open.
****S008****	Labeled file EOF1 trailer label contains invalid information that does not correspond to header label 1.	This file cannot be processed with standard labels.
****S009****	Labeled file is missing EOF trailer labels.	File cannot be processed as labeled.
****S010****	End of tape sensed on output file (unlabeled)	Close file with EOV and re-open after mounting new tape. Re-enter function to complete processing.
****S011****	Double file mark sensed on input file. Processing terminates.	Close input file and mount next volume. Re-enter function to complete processing.
****S012****	Invalid date	Re-enter date function with proper date.
****S013****	Labeled volume sequence number incorrect. This occurs after OPEN file is not opened.	Mount proper volume and re-open file.
****S014****	ZERO LENGTH block specified in OPEN file not opened	Re-open specifying proper block length.
****S015****	Block or record length specified is not a multiple of two. File is not opened.	Re-open specifying even block and record length. If either block or record length is odd, the data cannot be processed by the system.

3. Serious errors result from improper specification of valid parameters, tape errors, etc.

TABLE F-10. MAGNETIC TAPE UTILITY PROCESSOR SYSTEM ERRORS (Continued)

Warning Errors[4]		
Code	Description	Action
xxxW000xxx	Blocking not specified, but block size and record size have been specified differently in OPEN.	Open file with proper parameters, or continue.
xxxW001xxx	File count specified as zero.	Re-enter function with proper parameters or continue.
xxxW002xxx	Record count specified as zero.	Re-enter function with proper parameters or continue.
xxxW003xxx	Input and output record lengths have specified differently for copy.	Re-enter function with proper parameters or continue.

4. Warning messages indicate that parameters are possibly inconsistent or that a possible processing error has occurred.

TABLE F-11. ASSEMBLER ERRORS

Message	Description
llll**xx	Format for pass 1 error messages llll is the record number on which error occurred. xx is the type of error.
********xx	Format for pass 2 error messages xx is the type of error.
DS	Double defined symbol. A name in one of the following: The location field of a machine instruction or an ALF, NUM, or ADC pseudo instruction The address of an EAU, COM, DAT, EXT, BSS, or BZS pseudo instruction has been used again in one of the above fields.
EX	One of the following illegal expressions: No forward referencing of some symbolic operands No relocation of certain expression values A violation of relocation
IX	Illegal index register; specified by symbol other than Q, I, or B
LB	Numeric or symbolic label contains illegal character. Label is ignored.
OP	Illegal symbol in operation code field, or Illegal operation code terminator
OR	Numeric or symbolic operand in address expression contains illegal characters.
OV	Numeric value is greater than allowed.

TABLE F-11. ASSEMBLER ERRORS (Continued)

Message	Description
RG	Symbol other than A, Q, or M used in address field of inter-register instruction or same symbol used more than once, or Registers separated by other than a comma
RL	Violation of relocation, or Violation of a rule for instructions that require the expression value to be absolute or to have no forward referencing of symbolic operands
SO	Available storage for saving symbol name is exceeded; no more names may be defined.
UD	Undefined symbol in an address expression

TABLE F-12. FORTRAN EXECUTION ERRORS

Message	Description	Action
1 I/O RQST statement no. ffff	Error in format statement; illegal character in format statement	Program terminates.
2 I/O RQST statement no. ffff gggg	Illegal character in input field	Program terminates.
3 I/O RQST statement no. ffff gggg	Input data exceeds limits of 1700 word; exponent >39. ffff is the current decimal value of format statement pointer. gggg is the current decimal value of input field pointer.	Program terminates.
4 I/O RQST statement no. xx	Attempt to read on a write unit or write on a read unit	Program terminates.
5 I/O RQST statement no. xx	Read or write request after end-of-file has been read without first doing EOF check	Program terminates.
6 I/O RQST statement no. xx	Attempt to write EOF, to rewind, or to back-space any unit other than magnetic tape unit	Program terminates.
7 I/O RQST statement no. xx	Write attempted on magnetic tape with no write enable	To continue, press RETURN.
8 I/O RQST statement no. xx	Attempt to use logical unit number greater than 30	Program terminates.
9 I/O RQST statement no. xx	Backspace at loadpoint	Program terminates.

TABLE F-12. FORTRAN EXECUTION ERRORS (Continued)

Message	Description	Action
10 I/O RQST statement no. xx	End of magnetic tape sensed xx is the decimal unit number of device used improperly.	To continue, press RETURN.
11 I/O RQST statement no. xx	Illegal binary input; WRITE (u) is illegal with no list. xx is the decimal unit number of device used improperly.	Program terminates.
12 I/O RQST statement no. ffff	Illegally formatted input; more elements are given than are contained in input record.	Program terminates.
13 I/O RQST statement no. ffff	Illegal list; list is given but there are no conversion codes in the format statement. ffff is the current decimal value of format statement pointer.	Program terminates.
14 I/O RQST statement no. nn	File defined twice; more than one OPEN request is given for same file.	Program terminates.
15 I/O RQST statement no. nn	Parameter negative or zero; one of parameters in OPEN statement is negative or zero.	Program terminates.
16 I/O RQST statement no. nn	Sector address too large; starting or ending address exceeds 215-1.	Program terminates.
17 I/O RQST statement no. nn	File not defined; READ/WRITE request given for a file that was not defined by an OPEN statement.	Program terminates.
18 I/O RQST statement no. nn	Logical unit not a mass storage device	Program terminates.
19 I/O RQST statement no. nn	Record number in READ/WRITE request incorrect. Resulting sector address is out of the range of the file or is zero. nn is the decimal file number for mass storage device.	Program terminates.

TABLE F-13. DEVICE FAILURE ERRORS

Code	Error	Description
0	Time-out error	Failure to interrupt within allotted time (required TIMER package)
		Conversational Display Terminal: 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 next data word appeared. Use CU option to continue without processing lost record or abort the read option.
2	Alarm	Indicates an abnormal condition
3	Parity error	Repeat read request by typing RP in response to error message. Take CU option to continue processing (bad record will be ignored) or abort operation.
4	Checksum error	(FREAD binary) Sum of header word and data in a record did not balance to zero when added to checksum word.
		Attempt recovery by manually positioning the input medium to the beginning of last record. Repeat read request by typing RP in response to error message; otherwise, use the CU option.
5	Internal reject	I/O device did not send reply to computer within allotted time.
		Computer cannot communicate with device. Check hardware address switch and POWER ON switch. RP option may be used if problem has been corrected.
6	External reject	I/O device is not ready to perform specified request.
		Device is busy or not ready. If the device is not busy, check READY switch. Attempt to continue by typing RP.
7	Compare	Hardware problem. Compare error occurs when faulty signal is detected in area of the punch solenoid and echo amplifier circuits during an echo check. Attempt to reread the last record using the RP option or continue with error via the CU option.
8	Illegal Hollerith punch	Occurs when card reader has encountered punch sequence not complying with Hollerith-to-ASCII conversion table used by driver
		Software recovery allows user to locate the illegal punch by setting an ASCII? in buffer word for bad column. Select reply option to continue or abort job and correct mispunched cards.
9	Sequence error	Cards within a record are not in sequential order.
		Abort for read operation and restore sequential order to the record.
10	Non-negative record length	First word of formatted binary record is the complement of number of record within record. Word may be a negative number indicating that card read was not first card of record.
		Attempt recovery using procedure for checksum error (see code 4).
11	Read/write mode change	A switch from read or write mode
		If MODE switch is allowable, repeat request using RP option.

TABLE F-13. DEVICE FAILURE ERRORS (Continued)

Code	Error	Description
12	7/9 punch error	A 7/9 punch in column 1 was read when an FREAD ASCII request was specified. 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.
13	No write ring	Attempt was made to write on magnetic tape without write enabled or attempt was made to write on a file that was opened to read only. Insert write ring and use RP option.
14	Not ready	Device is not ready. Ready device and use RP option.
15		Not used
16	Controller seek error	Controller has failed to obtain file address selected during read, write, compare, or checkword operation. Usually an indication of a positioning error
17	Drive seek error	Cylinder positioner has moved beyond legal limits of device during load address, write, read, compare, checkword check, or write address function.
18	Address	Illegal file address was obtained from computer, or controller has advanced beyond limits of file storage.
19	Protect fault	Unprotected controller operation has attempted to write in a protected core location.
20	Checkword error	Controller logic has detected an incorrect checkword in data read from file storage during a read, compare, or checkword operation.
21		Not used.
22	Card output stacker full	Empty output hopper. Single-cycle cards from transport area into output stacker. Take last card in output hopper and put it into input hopper ahead of unread cards. Reload memory. Use RP option.
23	Card input hopper empty	If read operation is completed, use CU option; otherwise, supply more cards and take RP option.
24	Card feed failure	Read ready station does not contain a card after a feed cycle has occurred, and input hopper is not empty. Card feed failure error can occur as result of warped or damaged cards. If card reader can be made ready, tape RP option.
25	Card jam	Card transport problem has occurred.
26		Not enough file space available for this request to pseudo tape driver.
27		Not used
28	File error	No file assigned to this logical unit (pseudo tape driver)
29	Read error	Error occurred in reading mass storage resident driver.
30	Validation error	Frame punched does not compare with original data. Abort punch operation.

TABLE F-13. DEVICE FAILURE ERRORS (Continued)

Code	Error	Description
31	Short record	Attempt was made to write a record short than standard noise record length.
32		Not used
33	Line break	Line break occurred while attempting to input.
34	Data interrupt	Data interrupt occurred after reading 80 columns. This error indicates a hardware failure possibly due to improper card travel. Reread card (see recovery procedure for code 4).
35	End-of-operation	End-of-operation interrupt occurred prior to reading 80 columns. Continuous failure may indicate card slippage in feeding. Reread card as for code 4.
36		Reserved
37	Wrong address	Buffered data channel is using first word address other than address sent by buffered driver.
40	Repeated request due to error	Driver is attempting recovery.
41	Incomplete request	Request was not successfully completed. 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
45	Timing error	Occurred while drum was not busy
46	External reject	On output
47	External reject	On input
48	Controller address error	Controller address status not expected value
49	Drive address error	Drive address status not expected value
50	No ID	ID abort, no ID burst on PE tapes
51	Illegal density	Attempt to select illegal density
52	Power failure	Power failure
53	EOP error	EOP not set after interrupt
54	Data error	Data not set after interrupt
55	Bad status	Bad status, an indeterminate error occurred
56	Mass memory buffer expired	No more buffer space available (software buffer driver)
57	Buffer transfer error	Mass memory error on buffer transfer (software buffer driver)

TABLE F-13. DEVICE FAILURE ERRORS (Continued)

Code	Error	Description
58	Paper tape record error	System initializer paper tape reader driver has detected a record of illegal length.
59	PE lost data	Error in PE formatter that affected data transfer
60	Illegal motion	Motion request contains an illegal code (8-F_{16}).

INDEX

COMMENT SHEET

MANUAL TITLE	CONTROL DATA® REAL-TIME OPERATING SYSTEM VERSION 3 REFERENCE MANUAL		
PUBLICATION NO.	96769500	REVISION	A
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 to which your comment applies.

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